VOLTAR

Design Patterns

Apresentar brevemente os principais conceitos associados ao uso de Design Patterns

NESTE TÓPICO

Basicamente, o desenvolvimento de software possui dois caminhos a serem seguidos. No primeiro deles, muito por questões culturais, os desenvolvedores constroem seus aplicativos, sem avaliar, da maneira devida, quais os problemas que o software pode vir a ter, ou quando uma determinada solução pode ser reaproveitada, evitando ser desenvolvida duas vezes, e muitas vezes, replicando o que já foi desenvolvido, e não reaproveitando. De início, se uma parte do software que foi replicada tiver que ser alterada no futuro, será gasto muito tempo na atualização do software, e ainda, sem contar com o tempo gasto para realizar todos os testes novamente, e possíveis correções de erros. Apesar de ser mais simples, nem sempre esse modo é o melhor caminho a ser seguido.

No segundo modo, o desenvolvedor segue regras definidas pelos modelos de arquitetura, orientações definidas pelos frameworks e padrões de projeto. É um modo mais complexo de se desenvolver o software, mas ele é feito de modo mais organizado. Neste modo, costuma-se gastar mais tempo no planejamento, mas bem menos tempo nas outras atividades. Normalmente, o senso da necessidade de organizar o software vem com a experiência, pois com o passar do tempo, e tendo mais experiência no desenvolvimento, o desenvolvedor percebe que ele pode ter seu trabalho facilitado.

Neste tópico, estamos falando de soluções que se baseiam em modelos de programação, ou outros tipos de ferramentas, que permitam acelerar o desenvolvimento do software. Nesse contextos estão as APIs, os frameworks, ferramentas de modelagem e construção rápida do software. Usar essas ferramentas exige um período de estudo e aprendizado, e em muitas

situações isso ocorre em paralelo com o desenvolvimento do projeto. Isso pode desmotivar a adoção dessas ferramentas, principalmente entre os desenvolvedores mais novos, mas a médio e longo prazo, ele se mostra o melhor caminho a ser seguido. Tanto que há vários anos, várias comunidades de desenvolvedores vêem promovendo o uso de melhores práticas de programação, além de outros aspectos relacionados ao desenvolvimento ágil de aplicativos. Para se ter uma ideia, estima-se que no Brasil sejam gastos em torno de 10% do tempo com análise, 20% com programação, e o restante, em correção de erros de programação ou outras atividades de alteração do software.

Em países como Alemanha, onde os desenvolvedores são criados numa cultura diferente de trabalho, direcionada a realizar o desenvolvimento de um aplicativo seguindo modelos de arquitetura, padrões de projeto e métodos ágeis de desenvolvimento. Já estruturando a arquitetura e os componentes de software nas fases iniciais do desenvolvimento, e determinando vários pontos de redundância que podem ser evitados. Seguindo esse método de trabalho, estima-se que do tempo total do projeto, sejam gastos 60% com análise, 30% com programação e mais 10% com correção de erros de programação e alteração do software.

Particularmente, eu e alguns colegas de profissão dizíamos que esses percentuais representam a diferença entre o *"fazer e depois pensar no que foi feito"* e *"pensar em como deve ser feito, e depois fazer"*. Nos últimos anos, a tendência do mercado nacional está mudando para a segunda opção. Entretanto, em algumas situações, não há como escapar da primeira opção, mas isso não é motivo para não acertar o software quando houver oportunidade para isso.

Ser um desenvolvedor de software que segue a linha *"pensar em como deve ser feito, e depois fazer"* significa que se trata de uma pessoa que está disposta a seguir algumas regras de trabalho, e uma dessas regras é usar ao máximo possível **padrões de projetos**, que também são conhecidos como **design patterns**. Os livros *Padrões de Projeto* de Erich Gamma e *Utilizando UML e Padrões* de Craig Larman são dois clássicos sobre o assunto, explicando vários dos padrões existentes.

Os padrões de projetos são oriundos de boas práticas de programação, definidas por alguns desenvolvedores de software no início dos anos 1970. Em congressos, seminários, etc... , a troca de experiências entre eles fez com que fossem notados problemas que eram comuns a qualquer tipo de software, e o primeiro problema a ser atacado, foi a relação entre as interfaces de usuário, regras de negócio e bases de dados, e como uma dessas partes poderia ser modificada sem causar problemas às outras duas partes. Outra questão que foi analisada, foi a possibilidade de fazer um novo software, sem ser necessário fazê-lo desde o início.

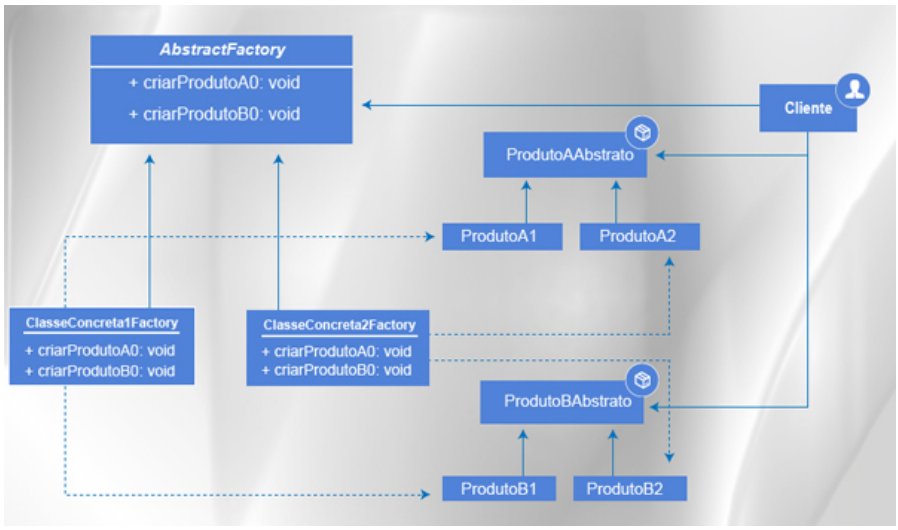

Figura 1 - Diagrama de classes em UML que ilustra a organização de classes para uma solução de software que usa um padrão de projetos.

Dessa troca de experiências, surgiu o **MVC (Model-View-Controller)**, um padrão de projeto que modela uma arquitetura de sofware para orientar os desenvolvedores de software a separar seu código-fonte em três partes principais: interação com o usuário (view), regras de negócio e rotinas de acesso às bases de dados (model) e algo que conecte essas duas partes (controller); Essas três partes serão totalmente independentes entre si, e se conectadas corretamente, formam um software totalmente funcional. A melhor analogia com isso seria um quebra-cabeças. Cada parte do software seria uma das peças, com um desenho específico que conecta uma peça a outras peças. Se todas estiverem corretamente conectadas, formam uma ilustração.

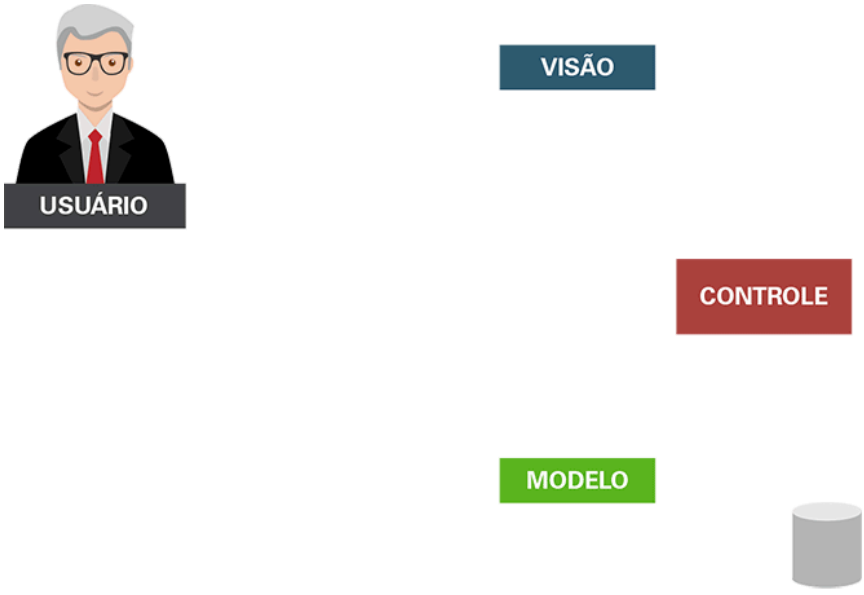

Figura 2 - Ilustração do funcionamento do padrão de projeto MVC.

O MVC surgiu com o objetivo de facilitar o trabalho em futuras atualizações de software, reduzindo o tempo gasto com desenvolvimento e com testes. Se o projeto for feito de acordo com o MVC, uma vez que um primeiro projeto esteja pronto, todas as suas partes podem ser reutilizadas em novos softwares. Assim, quando houver a necessidade de um novo desenvolvimento, a pessoa responsável pelo projeto pode verificar o que precisa ser feito, e decidir o que é possível reaproveitar das partes que já estão prontas, e o que deve ser feito, que são as partes novas. O tempo de desenvolvimento é reduzido ao conectar as partes novas às partes que foram reaproveitadas.

Analisando o que já vimos até o momento, essas partes podem ser chamadas de componentes de software, e definindo o que o MVC orienta a fazer, a comunicação entre esses componentes, de acordo com o MVC, ficará semelhante ao que mostra a Figura 2.

O MVC foi publicado em torno de 1975, e vem sendo modificado desde então, de acordo com as inovações tecnológicas. Entretanto, a idéia base ainda se mantém, e talvez ele seja o padrão de projeto mais usado entre os desenvolvedores de software. Usando o conceito de componentes de software de acordo com o modelo MVC, alguns componentes ficarão responsáveis pela interface de usuário, e nesse caso, qualquer coisa que tenha interação com o usuário pode ser entendido como interface de usuário, por exemplo: formulários para coleta de dados, relatórios, máquinas de cartão de débito ou crédito, telas, etc...

Quiz

Exercício Final

Design Patterns

INICIAR

Referências

GAMMA, E., JOHNSON, R. *Design Patterns*. Addisson-Wesley. 1994

LARMAN, C. *Utilizando UML e Padrões*. Bookman Cia Editora. 2007

Avalie este tópico

ANTERIOR
RMI, SOAP (Web Services) e REST

Índice

PRÓXIMO
Frameworks