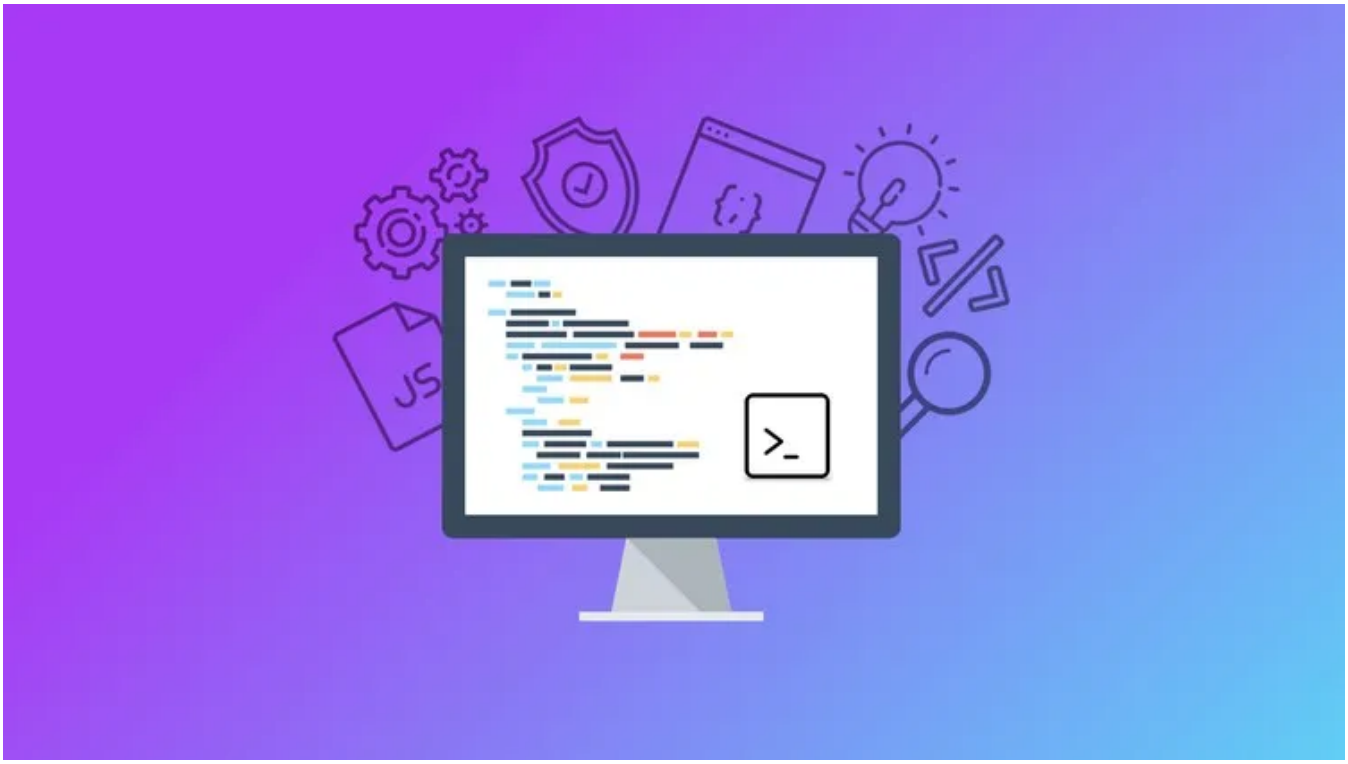
Leandro Vilas Boas (leandrovboas) Follow
Jan 11, 2019 · 4 min read



Padrões GRASP — Padrões de Atribuir Responsabilidades

Quando fui fazer o primeiro post sobre Designer Patterns, comentei com um amigo (Ricardo Araujo) que escreveria sobre o assunto, então ele me perguntou se eu conhecia o GRASP, e disse que seria interessante abordar o assunto, pois poucas pessoas conhecem, já que ele não é tão popular quanto o SOLID ou os padrões listados pelo GoF.

Então vamos ao que interessa...

Introdução

Assim como os padrões SOLID, o GRASP foi criado com o intuito de tornar o código mais flexível, facilitando a manutenção e a extensibilidade.

Os padrões GRASP (General Responsibility Assignment Software Patterns) consistem em uma série de princípios baseados em conceitos para atribuição de responsabilidades a classes e objetos na construção de bons softwares usando programação orientada a objetos.

Criação

Este conjunto de princípios foram publicados originalmente no livro "Applying UML and Patterns — An Introduction to Object-Oriented Analysis and Design and the Unified Process" (obra traduzida em português com o título "Utilizando UML e Padrões — Uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo") escrito pelo Craig Larman em 1997, contribuindo para a codificação de princípios de design de software, esse é um livro muito popular que contribuiu para a adoção generalizada do desenvolvimento orientado a objetos

Objetivo

O conceito de responsabilidade deve ser compreendido como as obrigações que um objeto possui quando se leva em conta o seu papel dentro de um determinado contexto. Além disso, é preciso considerar ainda as prováveis colaborações entre diferentes objetos.

A implementação de soluções OO considerando os conceitos de responsabilidade, papéis e colaborações fazem parte de uma abordagem conhecida como Responsibility-Driven Design ou simplesmente RDD.

Responsibility-Driven Design é uma técnica de desenvolvimento OO proposta no início dos anos 1990, como resultado do trabalho dos pesquisadores Rebecca Wirfs-Brock e Brian Wilkerson. Este paradigma está fundamentado nas ideias de responsabilidades, papéis e colaborações.

Partindo de práticas consagradas no desenvolvimento de aplicações orientados a objeto, procuram definir quais as obrigações dos diferentes tipos de objetos em uma aplicação, o GRASP disponibiliza uma catalogo de recomendações que procuram favorecer um conjunto de práticas visando um software bem estruturado.

A responsabilidade é definida como um contrato ou obrigação de uma classe e está relacionada ao comportamento. Existem 2 tipos de responsabilidades:

Saber:

- O conhecimento dos dados privados que o objeto em questão encapsula;
- Saber a respeito de outros objetos relacionados;
- Saber sobre as coisas que pode derivar ou calcular;

Fazer:

- A execução de ações que condizem com o papel desempenhado por tal objeto
- A criação de outros objetos dos quais a instância inicial depende;
- Controlando e coordenando atividades em outros objetos

Como os outros patterns conhecidos, os diferentes padrões GRASP não devem ser encarados como soluções pré-definidas para problemas específicos. Estes patterns devem ser compreendidos como princípios que auxiliam os desenvolvedores a projetar de uma forma bem estrutura aplicações orientadas a objetos.

Catálogo

Ao todo são nove os padrões GRASP, os quais serão discutidos em maiores detalhes nas próximas seções:

Information Expert (Especialista na Informação) — Determina quando devemos delegar a responsabilidade para um outro objeto que seja especialista naquele domínio.

Creator (Criador) — Determina qual classe deve ser responsável pela criação certos objetos.

Controller (Controlador) — Atribui a responsabilidade de lidar com os eventos do sistema para uma classe que representa a um cenário de caso de uso do sistema global.

Low Coupling (Baixo Acoplamento) — Determina que as classes não devem depender de objetos concretos e sim de abstrações, para permitir que haja mudanças sem impacto.

High Cohesion (Alta Coesão) — este princípio determina que as classes devem se focar apenas na sua responsabilidade.

Polymorphism (Polimorfismo) — As responsabilidades devem ser atribuídas a abstrações e não a objetos concretos, permitindo que eles possam variar conforme a necessidade.

Pure Fabrication (Pura Fabricação) — é uma classe que não representa nenhum conceito no domínio do problema, ela apenas funciona como uma classe prestadora de serviços, e é projetada para que possamos ter um baixo acoplamento e alta coesão no sistema.

Indirection (indireção) — este princípio ajuda a manter o baixo acoplamento, através de delegação de responsabilidades através de uma classe mediadora.

Protected Variations (Proteção contra Variações) — Protege o sistema com a variação de componentes, encapsulando o comportamento que realmente importa.

Conclusão

Todos esses padrões servem para a resolução de problemas comuns típicos de desenvolvimento de software orientado a objeto. Tais técnicas apenas documentam e normatizam as práticas já consolidadas, testadas e conhecidas no mercado.

Práticas de engenharia de software são relativamente desconhecidas ou pouco aplicadas pelos desenvolvedores e há uma suposição de que é tudo complicado ou muito bonito no papel, mas se você realmente entender como eles funcionam e quando aplicá-los, pode-se perceber que é fácil implementar e obter seus benefícios.