O princípio da responsabilidade única é uma regra?

Autores:
Guilherme Lorençato Guimarães Lamonato, 758665
Philipe Fonseca Bittencourt, 726579
Victoria de Martini de Souza, 759378

O que é o princípio da responsabilidade única(SRP)? “Uma classe deve ter somente uma razão para mudar”. Isso quer dizer que uma classe deve ter apenas uma motivo para ser modificada, em termos práticos, uma classe deve ser responsável por apenas uma coisa.

Esse princípio tende a evitar que uma classe seja uma faz tudo (God Object), em vez disso o código se torna algo mais fácil de manter, por desenvolvedores diferentes. Contudo essa prática acaba por criar uma grande quantidade de classes com responsabilidade única(exerce apenas uma funcionalidade), e dependendo do projeto não é a melhor solução, ou a solução esperada.

Portanto, para fugir das más práticas, comumente chamadas de “antipadrão”, é fundamental ter um documento de requisitos consolidado que crie discussões produtivas com o propósito de tomar decisões concretas e fundamentadas, evitando assim uma complexidade acidental.

Uma forma de verificar se o código segue o princípio da responsabilidade única é estudar cada método buscando um responsável/ator pela funcionalidade e separar as classes por funções que não irão conflitar em um cenário onde precise mudar o comportamento. Para aderir o padrão SRP no seu código é preciso que os envolvidos conheçam bem o domínio do projeto, o que reforça a necessidade de um documento de requisitos consistente.

Contudo, é possível concluir que o princípio da responsabilidade única é um padrão a ser seguido e não necessariamente uma regra, seu uso vai depender do projeto, dos atores envolvidos e principalmente do documento de requisitos e seus casos de uso.

- **Exemplo do conceito sendo aplicado erradamente**

```
1   class Venda{
2       public calculaPreco(){}
3       public calculaDesconto(){}
4       public atualizaCatalogo(){}
5   }
```

Na classe *Venda* que foi designada para vender um produto, nota-se a implementação dos métodos de preço, desconto e atualização do catálogo da venda do produto. Ao verificar o código, nota-se que esta classe apresenta mais de uma responsabilidade, o que se em um futuro houver alguma necessidade de alteração, seja na lógica de venda do produto ou cálculo de desconto, será necessário alterar esta classe pelo menos mais de uma vez, que é contra o princípio da responsabilidade única.

- **Exemplo anterior modificado para que o conceito agora seja aplicado corretamente**

```
1   class Produto{
2       float valor;
3       int id;
4   }
5
6   class Preco{
7       public calculaPreco()
8   }
9
10  class Desconto{
11      public calculaDesconto()
12  }
13
14  class Catalogo{
15      public atualizaCatalogo()
16  }
```

Neste exemplo, separamos as funções de cálculo da venda do produto em outras classes, para que quando for necessário alterar alguma propriedade do produto no preço, no desconto ou no catálogo a mudança seja mais intuitiva,

realizando a mudança apenas na classe respectiva preservando, assim, o princípio da responsabilidade única. Neste exemplo cada classe tem uma única responsabilidade.

Referências:

Texto:

- https://www.campuscode.com.br/conteudos/s-o-l-i-d-principio-da-responsabilidade-unica

Vídeos:

- Qual a melhor linguagem para programação orientada a objetos? *https://youtu.be/gbgV5jKZfTk*
- Princípios SOLID na programação orientada a objetos - Princípio da Responsabilidade Única. *https://youtu.be/wwg-gWTuB1o*
- Design especulativo - Princípios da programação orientada a objetos *https://youtu.be/alwkvSaODHc*