Notas sobre o capitulo 3 de Clean Code – by Lucas Trevizan

Funções

O capitulo aborda sobre como escrever bem funções, com um exemplo prático de um código com uma única função imensa e não clara em seu propósito e depois a mesma refatorada em pequenas linhas.

Pequenas

Bob relata que em todos seus anos de experiência, a grande conclusão sobre funções é que elas devem ser muito pequenas, mais precisamente, no máximo 20 linhas.

Faça apenas uma coisa

No primeiro exemplo da função da listagem 3.1 fazia muito mais que uma coisa, enquanto a listagem 3.3 faz apenas uma. Se uma função faz somente os passos que o nome da função descreve, então ela faz apenas uma coisa.

Seções dentro de funções

Um indício óbvio de que uma função faz mais de uma coisa é existir seções dentro dela, não da para dividir em seções uma função que faz apenas uma coisa.

Um nível de abstração por função

Uma maneira de saber se nossa função faz apenas uma coisa, é verificar se todas as instruções dentro da função estão no mesmo nível de abstração. Muitos níveis diferentes geram confusão, e isso pode causar a mistura do que é essencial com o que é detalhe.

Ler o código de cima para baixo: *regra decrescente*

Queremos ler o código de cima para baixo como uma narrativa, onde cada passo abaixo complementa a narrativa do passo acima. Essa regra é uma dica

de ouro para manter as funções fazendo apenas "uma coisa", fazer com que ela possa ser lida de cima para baixo como uma série de parágrafos a torna consistente.

Use nomes descritivos

Quanto menor e mais centralizada for a sua função, mais fácil será escolher o nome dela, nomes longos e descritivos não são problemas, e uma dica pra escolher um bom nome é ler a classe com vários possíveis nomes para a função em questão.

Parâmetros de funções

A quantidade ideal para parâmetros de uma função é 0, deve se evitar 3, e ter mais que 3 deve ser um caso extraordinário. Parâmetros são difíceis a partir de um ponto de vista de testes, imagine escrever um teste para certificar que mais de 3 parâmetros funcionem em diversas situações. Apenas um parâmetro de entrada é o melhor depois de 0 parâmetros.

Formas mônades comuns

Existem dois usos para 1 parâmetro de entrada que esperamos ver em uma função, fazendo uma pergunta sobre o parâmetro (usando um boolean), ou recebendo o parâmetro e transformando ele em outra coisa e retornando-o. Uma forma menos comum é tratando a função como um evento, nessa forma há um parâmetro de entrada mas nenhum de saída. Funções mônades que não seguem esses princípios devem ser evitadas.

Parâmetros lógicos

Esses parâmetros explicitam o fato de que a função faz mais de uma coisa, portanto devem ser evitados.

Funções díades

Uma função com dois parâmetros é mais difícil de ser lida do que uma monade. Há funções onde dois parâmetros são necessários, porém esses dois parâmetros são componentes de um único valor. Díades inevitavelmente serão

usadas e não são ruins, mas deve se considerar mecanismos para torna-lá uma mônade.

Funções tríades

Essas são ainda mais difíceis que díades, pense muito antes de usa-las.

Efeitos colaterais

Efeitos colaterais são mentiras, sua função diz que ela faz uma coisa, mas na verdade faz mais de uma e com isso gera acoplamentos e bugs. O livro apresenta um excelente exemplo de código para esse tema.

Parâmetros de saída

Se sua função precisa alterar o estado de algo, faça ela mudar o estado do objeto a que ela pertence.

Separação comando-consulta

A função deve fazer ou responder algo, nunca os dois.

Prefira exceções a retorno de erros

Auto explicativo, use exceções.

Como escrever funções como essa?

Não é possível criar uma função perfeita logo de cara, temos que escreve-la como uma "redação", fazemos o rascunho da ideia central, depois relemos e vemos o que pode ser mudado. Em outras palavras, prática constante.

Conclusão

Funções precisam ser curtas, bem nomeadas e bem organizadas. Queremos contar a estória de um sistema e funções nos ajudam nessa narração.