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

Funções

O capítulo aborda sobre como escrever funções de maneiras mais inteligentes. Com um exemplo prático de um código que tem 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. E nesse ponto eu concordo plenamente, quando estamos lidando com o código de outra pessoa, podemos acabar nos deparando com funções que usem 100 a 200 linhas (como eu já vi) e isso a torna muito difícil de se entender, pois geralmente temos muitas operações acontecendo ali dentro.

Faça apenas uma coisa *1.0*

Seguindo a ideia de que uma função deve ser pequena, então a conclusão de que uma função grande faz mais de uma coisa é óbvia. Portanto, uma das melhores maneiras de se ter funções mais legíveis é atribuindo o princípio da responsabilidade única a elas, isso eu já faço no meu dia a dia extraindo funções menores de dentro de uma função com escopo maior, esse pequeno ato facilita muito a legibilidade.

Seções dentro de funções *1.1*

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

Um nível de abstração por função *1.2*

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 1.3*

Parece óbvio falando assim, é como a maioria (todas né?) das pessoas leem, mas o sentido disso vai além, significa interpretar o código como uma narrativa, onde a linha de baixo complementa o que foi escrito na linha de cima.

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. Essa dica é ouro no dia a dia de

desenvolvimento, eu prático ela em toda nova classe/método que eu desenvolvo, sem economizar caracteres nos nomes, quando eu chego em um nome onde eu penso ***"é isso aí, acabei de descobrir o que esse método faz"*** eu sei que é um bom nome para utilizar.

Parâmetros de funções *2.0*

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.

Esse é um outro ponto que eu posso falar por experiência própria, diariamente eu lido com funções que tem mais de 4 a 5 parâmetros e de fato elas são pesadas para se testar e particularmente falando eu também as considero como impeditivo na hora de um entendimento fácil de um método.

Formas mônades comuns(Funções com 1 parâmetro) *2.1*

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 *2.2*

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

Funções díades (Funções com 2 parâmetros) *2.3*

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. De 3 parâmetros pra frente já estamos falando sobre maluquices que devem ser evitadas.

Parâmetros de saída *2.4*

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.

Conclusão

Funções precisam ser curtas, bem nomeadas e bem organizadas, de preferência com responsabilidades únicas e com menos parâmetros quanto forem possíveis. Queremos contar a estória de um sistema e funções nos ajudam nessa narrativa, eu diria até que só ajudar é um eufemismo.