Aluno: Sandro Matheus Ramos Turma: B

Resumo sobre o 2º e 3º capítulo do livro Clean Code de Robert Cecil Martin o “uncle bob”.

O livro aborda sobre a importância de ter um código limpo, mas o que vem a ser isso? Como abordado no livro, um código limpo é um código que quem fez teve vontade de fazer, um código que já te entrega no começo tudo o que se pretende fazer, um código tão claro que se tem uma previsibilidade sobre ele, todas as suas partes já entregam o que realizam, e não é preciso desencriptar o que a nele, não a truques nem sacadas, uma simples batida de olho já revela seu propósito. Por outro lado temos o código sujo, a pedra no sapato dos novos integrantes do time e o assassino da produtividade no projeto, uma variável sem um nome coerente como “var p1” ou uma função com 2.000 mil linhas e múltiplas funções onde para mudar o funcionamento de uma linha é preciso o entendimento de outras 1.999, no livro é abordado a importância de fazer um bom código, onde é levantado que a proporção de leitura e escrita em um código é de 10/1, com dados como estes é compreensivo que temos de dar muita atenção a isso no desenvolvimento do código.

**A importância dos nomes**

Colocamos nomes em variáveis, funções, métodos, arquivos, branches, ou seja, uma de nossas principais atribuições quando estamos codificando algo é a nomeação correta do que criamos, para assim posteriormente facilitar o entendimento. Dar nomes bons leva tempo mas economiza mais do que gasta, como já vimos na proporção de leitura e escrita, um exemplo que poderíamos ter como base seria o de um software de gerenciamento de estoque, onde temos uma variável chamada “Pv”, que significa produto vendido, porém seu propósito fica extremamente implícito e pode literalmente significar qualquer coisa, dificultando muito a quem fosse realizar a manutenção de nosso código ou inserir novas features, e poderíamos resolver esse problema de uma forma muito simples, apenas nomeando corretamente a variável para “produtoVendido”.

Com base no livro podemos absorver algumas dicas muito valiosas para quando estivermos prestes a nomear algo, eu dividi elas em 4 principais dicas, que se utilizarmos estaremos nomeando corretamente.

A primeira dica seria para resolver um problema inicial de falta de entendimento com relação ao que o programa deve fazer, sem essa clareza o desenvolvedor pode escolher nomes ruins, mas isso se resolve atualizando os nomes a medida que o projeto vai ficando mais claro.

Segunda dica seria a de evitar nomes que possam ter interpretações erradas, aqui tomamos o seguinte exemplo de uma variável declarada como String chamada “ListaProdutos”, o grande problema nisso é que dentro da programação temos o tipo List que trabalha com vetores e lendo um nome como esse no meio do código logo tendemos a interpretar como um possível vetor, sendo que na verdade ela é uma String, de tal modo que precisamos analisar a implementação para descobrir seu verdadeiro tipo, nisso também se enquadra a questão da nescidade de colocar o tipo no nome da variável, como por exemplo “nomeString”, tem como um nome não ser String? Ou outros como “AniversarioInt”, “vendasTable”, sendo que o tipo da variável no nome não muda nada.

Terceira dica, não abreviar os nomes, aqui tomamos como exemplo uma variável chamada “PCM”, caso visualizarmos essa variável no código e não soubéssemos o contexto dela no código nunca descobriríamos que seu propósito era de “Pagamentos Contas Mensais”, ou seja, jamais devemos abreviar nomes pois isso retira o seu sentido e dificulta o entendimento do código por novos integrantes do time.

Quarta dica, evitar nomes longos com microvariações, como o exemplo de “SaldoProdutos&VendasMes&Liquidez” e “SaldoProdutos&VendasMesReferido&Liquidez”, ambos são muito parecidos e na leitura do código poderiam ser facilmente confundidos.

**Funções**

Um programa bem escrito é feito com funções bem escritas, com relação a isso o livro aborda dois problemas principais, um com relação ao seu tamanho, e outro quanto a quantidade de funções/ações desempenhadas por determinada função.

O primeiro problema é relacionado ao tamanho correto da função, tomamos como base uma função chamada “GerenciamentoDeUsuario” com cerca de 1.000 linhas de código, caso haja um bug no meio disso tudo, seria uma grande dor de cabeça encontrá-lo, também com relação a leitura dessa função, uma função bem escrita o nome já entrega o que ela realiza, o que pode haver nessa função? Um cadastro, deletar, alterar os dados, não é possível saber pois tudo está desorganizado, também inserir novas features seria como pisar em ovos, alterar algo na função pode comprometer todo o programa, a solução abordada e recomendada pelo autor é a de desenvolvermos funções pequenas, assim isolamos todas as partes do código, facilitando a leitura, refatoração e detecção de bugs.

O nosso segundo problema provem do primeiro, tendo uma função gigantesca ela muito provavelmente será multitarefa, como no exemplo anterior da função “GerenciamentoDeUsuario”, se quisermos executar a função para alterar cadastros, ela também ira pedir um cadastro de usuário, como isso se quisermos executar algo específico nossa função fara diversas coisas de uma vez e não teremos muito controle sobre seu funcionamento, a solução é criarmos diversas funções especialistas, poderíamos quebrar nossa função do exemplo em “cadastroUsuario”, “validaUsuario”, “alteraUsuario”, “exluiUsuario”, isso facilita muito a leitura e não torna um mistério o que elas realizam no código e apenas batendo o olho já entendemos o que elas fazem, diferentemente da função “GerenciamentoDeUsuario” onde caso tivéssemos um problema na validação do usuário teríamos de vasculhar toda a implementação da função, já após a quebra em várias partes, iriamos direcionar nossa atenção somente a função que realizava aquela tarefa.

**Conclusão**

O que podemos levar destes dois capítulos e também abordado no início do livro, é que temos de dar atenção aos mínimos detalhes, pois se ignorarmos isso o todo estará comprometido e nunca seremos desenvolvedores realmente profissionais, temos de buscar o perfeccionismo no desenvolvimento do código para assim atendermos a nossa expectativa em realizar um bom trabalho e a do cliente em receber um bom software.