Resenha: *On the Criteria to Be Used in Decomposing Systems into Modules* (David L. Parnas, 1972)

O artigo de David L. Parnas, publicado em 1972, é um dos textos mais marcantes da área de engenharia de software. Nele, o autor discute como dividir um sistema em módulos de forma mais inteligente, pensando não apenas no funcionamento imediato, mas também na manutenção e evolução ao longo do tempo. Parnas critica a forma tradicional de modularização usada na época, que seguia quase sempre o fluxo de processamento dos programas, e apresenta uma nova ideia: organizar os módulos com base na ocultação de informações (*information hiding*).

Desde o começo, ele destaca que modularizar não é só separar o sistema em pedaços menores. É também uma forma de tornar o software mais fácil de entender, modificar e expandir. Isso traz três grandes vantagens: equipes diferentes podem trabalhar em paralelo sem atrapalhar umas às outras, mudanças em uma parte não exigem refazer todo o sistema e, por fim, cada módulo pode ser estudado e compreendido de forma isolada. É um raciocínio que ainda hoje faz todo sentido, mesmo em tempos de metodologias ágeis e arquiteturas modernas.

Para explicar suas ideias, Parnas usa um sistema chamado KWIC (Key Word in Context). Apesar de simples, ele serve como exemplo didático para mostrar diferentes formas de organizar um programa. O KWIC recebe várias linhas de texto, gera versões dessas linhas com rotações circulares das palavras e depois organiza tudo em ordem alfabética.

Parnas compara duas formas de estruturar o KWIC. A primeira é a mais comum: criar módulos seguindo o passo a passo do processamento (entrada de dados, geração dos shifts, ordenação, saída, etc.). Já a segunda é diferente: os módulos são criados para esconder certas decisões de projeto que podem mudar no futuro, como o formato dos dados, o método de armazenar linhas ou a forma de ordenar os textos.

Na comparação, fica claro que a segunda abordagem é mais robusta. Se, por exemplo, mudar o formato de entrada, na primeira decomposição praticamente todos os módulos precisariam ser alterados. Na segunda, a mudança ficaria restrita a apenas um módulo. Isso mostra o poder de separar as responsabilidades com foco em esconder detalhes internos, deixando o resto do sistema “protegido” de mudanças locais.

O conceito central do artigo é o de ocultação da informação. A ideia é simples: cada módulo deve ser responsável por esconder uma decisão de projeto que pode ser

modificada no futuro. Assim, os outros módulos só interagem com ele por meio de uma interface clara, sem precisar saber como as coisas são feitas internamente.

Parnas dá vários exemplos de onde isso faz sentido: estruturas de dados, sequências de processamento, formatos de blocos de controle e até convenções de ordenação de caracteres. Em todos esses casos, se esses detalhes estiverem escondidos dentro de um módulo específico, o sistema fica muito mais fácil de manter. Hoje, esse raciocínio é bem parecido com o que vemos em conceitos de encapsulamento na programação orientada a objetos e em arquiteturas baseadas em serviços.

Parnas também reconhece que usar *information hiding* pode trazer algum custo de desempenho, já que aumenta as chamadas entre módulos. No entanto, ele sugere formas de contornar esse problema, como o uso de montadores ou ferramentas que otimizem a implementação final sem sacrificar a clareza no design. Para ele, eficiência é importante, mas não deve vir à custa da manutenibilidade. Essa visão também se conecta ao que acontece hoje, quando muitas vezes aceitamos uma perda mínima de performance em troca de um software muito mais flexível.

Mesmo sendo um texto de 1972, o artigo continua atual. Muitos sistemas ainda sofrem porque foram divididos de forma “errada”, seguindo o fluxo de processamento em vez de pensar na evolução futura. O princípio de ocultação da informação continua fundamental, e vemos sua influência em praticamente todas as práticas modernas de design de software.

Além disso, Parnas mostra que modularizar bem não só ajuda na manutenção, mas também no reuso. Ele cita o caso de compiladores e interpretadores, em que a mesma decomposição por *information hiding* serviu para ambos os cenários, economizando esforço e aumentando a reutilização do código.

A principal mensagem do artigo é clara: não devemos modularizar sistemas apenas com base no fluxo de execução. O melhor critério é separar módulos de forma que cada um esconda decisões que podem mudar. Isso garante independência, clareza e flexibilidade.

Parnas, com esse texto, não só trouxe uma solução prática, mas também ajudou a consolidar a engenharia de software como disciplina. Seu raciocínio mostra que desenvolver software não é apenas escrever código para funcionar hoje, mas também projetar algo que consiga acompanhar mudanças e sobreviver no longo prazo. É por isso que esse artigo, escrito há mais de cinquenta anos, ainda é tão citado e estudado.