O artigo “On the Criteria to Be Used in Decomposing Systems into Modules” de David Parnas questiona a forma tradicional de dividir sistemas em módulos. Até então, a prática mais comum era decompor programas com base em passos do processamento, quase como uma tradução direta de um fluxograma para código. Isso gerava módulos dependentes de representações internas compartilhadas, dificultando mudanças, entendimento e evolução do sistema.

Parnas propõe uma abordagem diferente: a decomposição baseada em ocultamento de informações (*information hiding*). A ideia é que cada módulo deve esconder uma decisão de projeto suscetível a mudanças (por exemplo, formato de entrada, forma de armazenar dados em memória, organização de tabelas). Assim, se uma dessas decisões for alterada no futuro, apenas o módulo responsável precisaria ser modificado, preservando a estabilidade dos outros. Essa visão antecipa diretamente conceitos que hoje são básicos em Engenharia de Software, como baixo acoplamento e alta coesão, princípios de encapsulamento e a importância de projetar para manutenção e flexibilidade.

No artigo, Parnas exemplifica essa diferença com o problema do KWIC index. Ele mostra duas modularizações: a convencional, baseada em fases de processamento (entrada, circular shift, ordenação, saída), e a alternativa, baseada em ocultar estruturas e representações internas (um módulo para armazenamento de linhas, outro para circular shift, outro para ordenação, etc.). A segunda modularização se mostrou mais flexível, permitindo mudanças localizadas e maior independência entre equipes de desenvolvimento.

A análise de Parnas também chama atenção para outro ponto importante na Engenharia de Software: o código em execução pode ser igual em ambas as decomposições, mas a estrutura de projeto muda completamente a forma como o sistema pode ser entendido, mantido e evoluído. Ou seja, modularização não é apenas sobre eficiência de execução, mas sobre eficiência no ciclo de vida do software.

Esse trabalho é diretamente conectado com práticas modernas como design orientado a objetos, arquitetura de software baseada em componentes, programação orientada a serviços e até microservices. Todos esses paradigmas buscam justamente isolar decisões, garantir interfaces bem definidas e reduzir dependências. Além disso, está alinhado com princípios posteriores como os do SOLID (especialmente o *Single Responsibility Principle* e o *Open-Closed Principle*).

Em resumo, o artigo de Parnas mostra que pensar a modularização apenas como “dividir o programa em funções” é limitado. O mais importante é definir módulos em torno de decisões de projeto que podem mudar. Esse é um princípio central da Engenharia de Software moderna, que busca criar sistemas não apenas funcionais, mas também compreensíveis, evolutivos e gerenciáveis ao longo do tempo.