Resenha “On the Criteria To Be Used in Decomposing Systems into Modules” - Pedro Henrique Maia Alves

O artigo começa introduzindo a importância da modularização no desenvolvimento de software e questiona os critérios usados para dividir sistemas em módulos. David Parnas argumenta que, tradicionalmente, os sistemas eram decompostos em função do fluxo de processamento ou das etapas de execução, mas essa abordagem não favorece a manutenção nem a evolução do software. Ele propõe, em contrapartida, a ideia de dividir o sistema segundo a ocultação de informações (*information hiding*), ou seja, cada módulo deve esconder suas decisões de projeto internas e expor apenas uma interface estável. Dessa forma, a estrutura modular se torna mais resiliente a mudanças, permitindo que partes internas sejam alteradas sem impactar o resto do sistema.

Por que este critério é necessário? O artigo defende que, ao projetar módulos, não se deve focar em *como* o sistema executa as operações, mas em *quais decisões de projeto podem mudar* no futuro. Por exemplo, se o formato de armazenamento de dados é algo suscetível a alteração, essa responsabilidade deve ficar isolada em um módulo específico. Assim, a evolução do software é facilitada, já que apenas um ponto precisa ser modificado. Em contraste, a decomposição funcional tradicional tende a espalhar esse tipo de dependência em várias partes do sistema, tornando a manutenção custosa e arriscada.

Algumas características perceptíveis em sistemas bem modularizados, segundo o artigo, são:

- **Ocultação de informações** – Cada módulo guarda detalhes de implementação que não devem ser visíveis a outros módulos.

- **Interfaces estáveis e bem definidas** – A comunicação entre módulos ocorre por contratos claros, reduzindo dependências desnecessárias.

- **Baixo acoplamento** – Mudanças em um módulo não afetam outros, pois as dependências são minimizadas.

- **Alta coesão** – Cada módulo tem uma responsabilidade bem delimitada e consistente.

- **Independência na manutenção** – Desenvolvedores podem trabalhar em módulos diferentes sem interferir no trabalho dos outros.

- **Maior reutilização** – Módulos podem ser aplicados em outros projetos, pois não carregam dependências ocultas.

No artigo, Parnas ilustra sua proposta comparando dois projetos distintos para resolver o mesmo problema: um que segue a decomposição funcional tradicional e outro que segue o princípio da ocultação de informações. O segundo modelo mostra-se muito mais vantajoso, pois localiza mudanças em pontos específicos e evita que toda a base de código precise ser revisada quando decisões de projeto são revistas. O princípio de *information hiding* acaba se tornando o precursor direto dos padrões de design e das boas práticas modernas de engenharia de software.

Algumas estratégias propostas para lidar com a modularização incluem projetar módulos que escondam as decisões mais suscetíveis a mudanças, definir interfaces estáveis que não exponham detalhes internos e favorecer a independência dos módulos sempre que possível. Além disso, a documentação das interfaces é essencial para permitir que desenvolvedores compreendam o comportamento esperado de cada módulo sem conhecer sua implementação.

O artigo conclui que a verdadeira utilidade da modularização não é apenas organizar o código em pedaços menores, mas tornar o software mais fácil de modificar, estender e manter ao longo do tempo. Em vez de pensar na sequência de operações, o projetista deve pensar em quais aspectos do sistema devem permanecer ocultos e quais devem ser revelados. Essa mudança de paradigma foi crucial para a evolução da engenharia de software, servindo como base para princípios modernos como *encapsulamento* e *separação de preocupações*.

No dia a dia do mercado, acredito que aplicar o princípio da ocultação de informações e manter módulos com responsabilidades claras é fundamental para garantir a longevidade de um sistema. Projetos grandes e em constante evolução se beneficiam diretamente dessa abordagem, pois equipes diferentes conseguem trabalhar em paralelo sem comprometer a integridade do software. Assim, reforço que o artigo de Parnas continua extremamente atual e essencial para qualquer desenvolvedor ou arquiteto de software.