O artigo de D.L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules" (1972), é um texto fundamental na história da engenharia de software que questiona a abordagem convencional de modularização e propõe uma alternativa mais robusta. O autor argumenta que a eficácia da modularização depende dos critérios usados para dividir um sistema em módulos.

#### **Modularização Convencional vs. Abordagem de Ocultação de Informação**

O artigo apresenta um problema de design de um sistema de indexação KWIC (Key Word In Context) para ilustrar duas abordagens de modularização.

1. **Modularização 1 (Convencional):** Esta abordagem divide o sistema em módulos com base nas etapas de um fluxograma. Os módulos são: Input, Circular Shift, Alphabetizing, Output e Master Control. Parnas aponta que essa é a abordagem mais comum e que a maior parte do treinamento de programadores a promove. No entanto, essa decomposição tem sérias desvantagens. Por exemplo, se houver uma mudança na forma como os dados são armazenados (por exemplo, guardar todas as linhas na memória principal ou mudar o formato de empacotamento de caracteres), quase todos os módulos seriam afetados. Além disso, a compreensão de um módulo depende do entendimento de outros, tornando o sistema compreensível apenas como um todo. As interfaces entre os módulos são complexas, o que dificulta o desenvolvimento independente.
2. **Modularização 2 (Não Convencional):** Parnas propõe uma abordagem alternativa usando o princípio de **"ocultação de informação"** (information hiding). Nessa metodologia, cada módulo é projetado para esconder uma decisão de design complexa ou que é provável de mudar. Os módulos não correspondem mais a fases no processamento. Por exemplo, o módulo Line Storage (Armazenamento de Linhas) esconde os detalhes de como as linhas são armazenadas. Mudar a forma de armazenamento dos caracteres (por exemplo, de 4 caracteres por palavra para 1 por palavra) seria uma alteração confinada exclusivamente a este módulo, sem afetar o resto do sistema. As interfaces nesta abordagem são mais abstratas e simples (principalmente nomes de funções e parâmetros), permitindo o desenvolvimento independente mais cedo. O autor classifica a omissão de esconder a ordem das shiftrs circulares como um "erro de design" na decomposição 2, pois essa informação desnecessária restringe a flexibilidade.

#### **Principais Vantagens e Considerações**

O artigo destaca os seguintes benefícios da abordagem de ocultação de informação:

* **Flexibilidade e Capacidade de Mudança:** Alterações em decisões de design, como o formato de entrada ou a estrutura de dados, são contidas em um único módulo, reduzindo o impacto no sistema como um todo.
* **Desenvolvimento Independente:** As interfaces simples e bem definidas permitem que as equipes trabalhem nos módulos de forma mais autônoma e iniciem o desenvolvimento mais cedo.
* **Compreensibilidade:** Cada módulo pode ser entendido isoladamente, sem a necessidade de compreender a lógica interna de outros módulos, tornando o sistema mais fácil de ser assimilado.

#### **Eficiência e Implementação**

Parnas reconhece que a decomposição baseada em ocultação de informação pode, à primeira vista, parecer menos eficiente devido ao excesso de chamadas de procedimentos. Para mitigar essa desvantagem, ele sugere uma abordagem de implementação onde as funções são inseridas diretamente no código por um montador, em vez de serem chamadas como sub-rotinas tradicionais. Essa técnica permite combinar a eficiência do código final com os benefícios de design da decomposição. Ele conclui que um módulo não deve ser, por definição, uma ou mais sub-rotinas, mas sim uma coleção de código que esconde uma decisão de design.

#### **Estrutura Hierárquica**

Parnas também aborda a diferença entre uma "decomposição limpa" (clean decomposition) e uma "estrutura hierárquica". Ele afirma que, embora desejáveis, são propriedades independentes. A abordagem de ocultação de informação, no entanto, tende a produzir uma estrutura hierárquica mais clara, onde os módulos de "alto nível" usam os serviços dos módulos de "baixo nível" sem depender de sua implementação interna. Essa hierarquia permite que partes do sistema sejam reutilizadas ou podadas para outros projetos, o que não seria possível se os módulos de baixo nível dependessem dos de alto nível.