Aluno : Thomás Ramos Oliveira

On the Criteria to be Used in Decomposing Systems into Modules

O artigo de David L. Parnas é um dos mais importantes quando se fala em engenharia de software, porque ele mudou a forma de pensar a modularização. Antes, a maioria dos programadores dividia um sistema em módulos de acordo com as etapas de execução: um módulo para entrada, outro para processamento, outro para saída, e assim por diante. Essa abordagem parecia organizada, mas tinha um problema sério: qualquer mudança pequena em uma parte do sistema acabava impactando vários módulos ao mesmo tempo. Isso deixava o software difícil de manter, de entender e de evoluir.

Parnas percebeu isso e propôs um critério novo, chamado **ocultamento de informações** (*information hiding*). A ideia é que cada módulo deve ser feito para esconder certas decisões de projeto que provavelmente vão mudar no futuro. Assim, quando o sistema precisar de ajustes, basta modificar apenas o módulo relacionado àquela decisão, sem que o resto seja afetado. Essa simples mudança de visão trouxe muito mais flexibilidade e tornou os sistemas mais fáceis de manter ao longo do tempo.

O exemplo do KWIC

Para explicar sua ideia, Parnas usa o exemplo do **KWIC (Key Word in Context)**. Esse sistema pega várias linhas de texto, cria versões circulares de cada linha (girando as palavras de posição) e depois organiza tudo em ordem alfabética. Ele mostra duas formas diferentes de dividir esse sistema em módulos.

Na **primeira forma**, que é a mais tradicional, os módulos são separados em etapas: um para ler os dados, outro para gerar os deslocamentos, outro para ordenar e outro para imprimir o resultado. Essa organização é simples, mas o problema é que qualquer alteração, como mudar o formato da entrada ou o jeito de armazenar as linhas, obriga a modificar vários módulos.

Na **segunda forma**, Parnas aplica sua ideia de ocultamento de informações. Os módulos são organizados não pelas etapas do processo, mas pelas decisões de projeto que podem mudar. Então há um módulo só para cuidar do armazenamento das linhas, outro só para gerar os deslocamentos circulares, outro apenas para organizar em ordem alfabética, e assim por diante. Se o modo de armazenamento precisar ser alterado, por exemplo, apenas o módulo responsável por isso será modificado, e o restante do sistema continua igual.

Essa comparação deixa claro como a segunda abordagem é mais vantajosa: ela deixa os módulos mais independentes, facilita a manutenção e diminui o impacto de mudanças.

Benefícios da abordagem de Parnas

A proposta de Parnas traz benefícios muito relevantes:

1. **Mais rapidez no desenvolvimento** – como os módulos são mais independentes, equipes diferentes podem trabalhar em cada um sem precisar se preocupar tanto com os outros. Isso reduz o tempo de comunicação e acelera o processo de criação.
2. **Maior flexibilidade do sistema** – mudanças acontecem o tempo todo em software. A ideia de esconder as decisões que podem mudar permite que um sistema se adapte de forma mais fácil, sem precisar ser refeito.
3. **Facilidade para entender** – quando cada módulo tem uma função bem definida e independente, fica mais simples para um programador estudar o sistema. Ele pode entender apenas uma parte sem precisar conhecer todos os detalhes das outras.
4. **Reutilização de módulos** – como os módulos são independentes, eles podem ser reaproveitados em outros projetos. Isso economiza tempo e aumenta a qualidade do software.

Questão da eficiência

Um ponto que Parnas também aborda é a eficiência. Se cada módulo for feito como uma função isolada, pode parecer que o sistema fica mais lento por causa das várias chamadas entre módulos. Mas ele explica que, com boas técnicas de implementação, esse problema pode ser resolvido. Em outras palavras, não é necessário escolher entre flexibilidade e desempenho; é possível ter os dois se o sistema for bem planejado.

Hierarquia e modularização

O autor também comenta sobre hierarquia. Um sistema pode ser organizado de forma hierárquica, onde módulos de níveis mais baixos dão suporte aos de níveis mais altos. Isso ajuda na reutilização de partes do código. No entanto, Parnas deixa claro que **ter hierarquia não é suficiente** para garantir um bom projeto. O essencial é pensar em como os módulos escondem decisões que podem mudar. Ou seja, a hierarquia é útil, mas não substitui a ideia do ocultamento de informações.

Conclusão

O artigo de Parnas mostra que modularizar não é apenas dividir o sistema em pedaços, mas dividir de forma inteligente. A proposta de organizar os módulos com base no ocultamento de informações mudou a forma como os programadores passaram a pensar sobre software. Em vez de apenas seguir a lógica do fluxo de execução, o foco passou a ser na **manutenção e evolução futura do sistema**.

Essa visão ainda é extremamente atual, mesmo décadas depois. Projetos modernos continuam enfrentando o mesmo desafio: como criar sistemas grandes que sejam fáceis de entender, modificar e manter. O artigo de Parnas traz uma resposta clara: modularizar de modo que cada parte esconda suas decisões internas, tornando o sistema mais flexível, mais organizado e preparado para mudanças inevitáveis ao longo do tempo.

No fim, a lição mais importante é que a modularização deve pensar não apenas no presente, mas principalmente no futuro do software. E é justamente por isso que esse artigo continua sendo tão estudado e citado até hoje.