ALUNO: João Gabriel Santos Andrade Almeida

Este é o aclamado estudo de David Parnas sobre modularização, que transformou a maneira como encaramos a divisão de sistemas em partes menores. Parnas questiona a forma usual de pensar a modularização e propõe um critério diferente, que se tornou essencial na engenharia de software.

O estudo começa explicando as vantagens esperadas da programação modular: gestão (desenvolvimento mais rápido com equipes atuando de forma independente), adaptabilidade (capacidade de fazer mudanças significativas em um módulo sem afetar os demais) e facilidade de entendimento (estudar o sistema um módulo de cada vez). Parnas define módulo não como um subprograma, mas como uma atribuição de responsabilidade - decisões de projeto que devem ser tomadas antes que o trabalho independente comece.

Para ilustrar suas ideias, Parnas usa um exemplo simples de propósito: um sistema de índice KWIC (Key Word In Context) que recebe linhas de texto, gera todas as rotações circulares possíveis de cada linha e cria uma listagem em ordem alfabética. Mesmo sendo um sistema pequeno que um bom programador faria em uma ou duas semanas, ele o trata como se fosse um grande projeto para demonstrar os princípios.

A primeira modularização segue a abordagem tradicional, dividindo o sistema com base no fluxo de processamento: módulo de entrada (lê e armazena dados), módulo de rotação circular (cria índice das rotações), módulo de alfabetização (ordena as rotações), módulo de saída (formata a listagem) e controle mestre (sequencia os outros módulos). É exatamente o que você faria se seguisse um fluxograma - cada etapa se torna um módulo.

A segunda modularização usa uma abordagem totalmente diferente baseada em "information hiding" (ocultação de informação). Os módulos não correspondem mais às etapas do processamento, mas a decisões de projeto que devem ser escondidas: armazenamento de linhas (esconde como as linhas são armazenadas internamente), entrada (usa o módulo de armazenamento), rotação circular (cria a ilusão de que todas as rotações existem sem necessariamente armazená-las), alfabetização (esconde o método de ordenação), saída e controle mestre.

A diferença fica evidente quando você considera mudanças prováveis: formato de entrada, decisão de manter tudo na memória versus usar disco, empacotamento de caracteres, se calcular rotações na hora ou pré-computar e estratégia de alfabetização. Na primeira modularização, mudanças como o formato de armazenamento afetam todos os módulos, pois todos precisam conhecer a estrutura interna dos dados. Na segunda, essas mudanças ficam restritas a módulos específicos.

Por exemplo, imagine que você queira modificar a maneira como as linhas são guardadas (talvez descomprimir caracteres ou usar o disco em vez da memória). Na

primeira forma, todos os blocos seriam afetados, já que todos acessam diretamente a forma dos dados. Já na segunda, só o bloco que guarda as linhas precisaria ser alterado; os outros blocos só conhecem as funções públicas, como CHAR(linha, palavra, caractere), e não precisam saber como isso funciona internamente.

A criação separada também se torna mais fácil com a segunda maneira. Na primeira, as ligações entre os blocos são formatos complicados de tabelas e estruturas de dados que precisam ser muito bem pensados em conjunto. Na segunda, as ligações são mais gerais: nomes de funções, tipos e quantidades de parâmetros, decisões mais simples que permitem que o trabalho separado comece bem antes.

Pensando em clareza, na primeira divisão, é preciso entender um pouco sobre todos os outros blocos para entender qualquer um deles, já que as formas dos dados são divididas e têm limites que só fazem sentido se você conhecer os processos dos outros blocos. Na segunda, cada bloco pode ser entendido sozinho.

Parnas mostra o ponto chave: a primeira divisão foi feita pensando em transformar cada passo principal do processo em um bloco, seguindo um tipo de mapa. A segunda usou a ideia de "esconder informações" como guia: cada bloco esconde uma escolha de projeto específica dos outros. Isso muda tudo, pois os blocos não estão mais ligados a momentos específicos do processo.

Ele diz que melhorou o bloco de rotação circular depois de pensar bem; mostrar a ordem das rotações dava informações desnecessárias. Bastaria garantir que todas as rotações existissem, sem cópias, com uma função para achar a linha original. Ditar a ordem limitava demais os tipos de sistema que podiam ser criados.

Parnas dá alguns exemplos de como dividir baseado em esconder informações: formas de dados com seus jeitos de acesso devem formar um único bloco; ordens de chamada de funções devem estar no mesmo bloco que a função; formatos de partes de controle em sistemas devem ser escondidos; códigos de letras e ordens de palavras devem ser escondidos; ordens de processo devem ser escondidas sempre que possível.

Uma preocupação válida é a eficiência. A segunda divisão pode ser menos eficiente se feita da forma comum, com cada função sendo um processo com uma sequência de chamada grande. Haveria muitas chamadas por causa da troca constante entre os blocos. Para resolver isso, Parnas sugere fazer diferente, com funções colocadas no código pelo montador, ou transferências especiais e rápidas. Precisa de ferramentas que deixem escrever como se fossem sub-rotinas, mas montadas como a implementação precisa.

Outro caso notável apresentado é o de um conversor para algoritmos de Markov, no qual a fragmentação pautada no ocultamento de informação mostrou-se eficaz tanto para compiladores quanto para intérpretes da mesma linguagem. Aspectos como a representação de registradores, algoritmos de pesquisa e a interpretação de

regras constituíam módulos presentes em ambos os tipos de conversores. Uma divisão seguindo abordagens tradicionais (analisador sintático, criador de código, rotinas de execução) não teria viabilizado o uso para ambos.

Parnas estabelece uma distinção entre fragmentação clara e a organização hierárquica. Na segunda forma de modularização, é possível identificar uma hierarquia no sentido proposto por Dijkstra: a tabela de símbolos no primeiro nível, o armazenamento de linhas no segundo nível, a entrada e a rotação circular necessitando de armazenamento, a saída e a organização alfabética exigindo rotação circular. A estrutura hierárquica permite "remover" os níveis mais altos e ainda obter um resultado funcional. A tabela de símbolos, por exemplo, pode ser aplicada em outros contextos, e o armazenamento de linhas poderia ser a base de um sistema de perguntas e respostas.

Contudo, é possível ter uma fragmentação clara sem hierarquia, assim como uma hierarquia sem fragmentação clara. São atributos independentes e valiosos na estrutura de um sistema.