Resenha: O Critério Decisivo para Modularização de D.L. Parnas

Em seu influente artigo de 1972, "On the Criteria To Be Used in Decomposing Systems into Modules", David Lorge Parnas oferece uma solução duradoura para uma questão central da engenharia de software. Publicado em um período conhecido como a "crise do software", onde projetos frequentemente excediam prazos e orçamentos, o trabalho de Parnas buscou substituir a intuição por um princípio de design rigoroso. Ele argumenta que a prática comum de modularizar com base no fluxo de processo é fundamentalmente falha e propõe, em seu lugar, o critério do **ocultamento de informação** (*information hiding*) como o caminho para criar software flexível e de fácil manutenção. Para provar seu ponto, Parnas utiliza um exemplo claro, um sistema de índice KWIC (Keyword in Context), para contrastar as duas abordagens.

A **abordagem convencional**, criticada por Parnas, decompõe o sistema seguindo as etapas de um fluxograma. Essa lógica é intuitiva, pois espelha como descreveríamos o processo verbalmente. O resultado são módulos como *Input*, que lê e armazena os dados; *Circular Shift*, que processa os dados para criar um índice de permutações; *Alphabetizing*, que ordena esse índice; e *Output*, que o imprime. O problema fundamental desta abordagem reside nas interfaces entre os módulos, que são as próprias **estruturas de dados compartilhadas**. O formato em que os dados são armazenados na memória, por exemplo, precisa ser conhecido por quase todos os módulos. Isso cria um sistema de **alto acoplamento**, onde os componentes são interdependentes. Uma mudança em uma decisão de design — como a compactação de caracteres para economizar memória — exige modificações em cascata por todo o código, tornando a evolução do software cara, demorada e propensa a erros.

Em contrapartida, a **proposta de Parnas** é modularizar com base na ocultação de "segredos", ou seja, decisões de design que são complexas ou prováveis de mudar. Nesta abordagem, cada módulo encapsula uma decisão específica, protegendo o resto do sistema de sua complexidade. Um módulo de *Line Storage*, por exemplo, esconde todos os detalhes de como as linhas são fisicamente armazenadas (se estão em memória ou em disco, compactadas ou não), expondo apenas funções abstratas para acessar os dados. Um módulo de *Alphabetizer* esconde não apenas o algoritmo de ordenação, mas também *quando* ela ocorre (de uma só vez ou de forma incremental). As interfaces, neste caso, não são dados, mas sim **funções abstratas**. Os módulos interagem através de um contrato estável, resultando em **baixo acoplamento** e permitindo que a implementação interna de um módulo evolua sem impactar os outros.

A superioridade do critério de Parnas é demonstrada de forma inequívoca ao se considerar a **flexibilidade** do sistema diante de mudanças, o benefício mais crítico

para a longevidade de um projeto. Se um desenvolvedor decidir alterar a forma de armazenamento de dados para otimizar o desempenho, no modelo convencional, a tarefa seria imensa. Todos os módulos que tocam nesses dados precisariam ser identificados, analisados e reescritos. No modelo de Parnas, a mudança ficaria **inteiramente contida** dentro do módulo de *Line Storage*. Nenhum outro módulo seria afetado, pois a sua interface funcional permaneceria a mesma. A manutenção se torna localizada, previsível e segura. Parnas destaca que essa abordagem não só aumenta a flexibilidade, mas também facilita o **desenvolvimento paralelo** — pois as equipes podem trabalhar em módulos diferentes com base em interfaces estáveis — e melhora a **compreensibilidade**, já que um módulo pode ser entendido apenas por sua interface, sem a necessidade de mergulhar em suas dependências.

O legado desta ideia é imenso e define muito do que hoje consideramos um bom design de software. O princípio do ocultamento de informação é a base conceitual para o **encapsulamento** na Programação Orientada a Objetos, para o design de APIs modernas, para a arquitetura em camadas e até mesmo para a filosofia por trás dos microserviços, onde cada serviço esconde sua lógica e seu modelo de dados. Parnas ensinou uma lição atemporal: um bom design modular não se trata de espelhar o fluxo de trabalho, mas de construir barreiras inteligentes em torno da incerteza e da complexidade, garantindo que o software do futuro possa ser construído sobre as fundações sólidas do presente.