A arquitetura hexagonal, proposta por Alistair Cockburn, representa uma das ideias mais elegantes e duradouras da engenharia de software moderna. Em seu artigo *Hexagonal Architecture (Ports and Adapters)*, Cockburn parte de uma constatação simples, mas profunda: os sistemas de software se tornam frágeis quando a lógica central de negócio — o "coração" da aplicação — fica amarrada a tecnologias externas como bancos de dados, interfaces gráficas, frameworks web ou APIs de terceiros. Essa dependência excessiva cria um tipo de rigidez estrutural que dificulta tanto a evolução do software quanto a sua testabilidade. O autor propõe, portanto, um novo modo de pensar o design: separar o domínio da aplicação (as regras de negócio) das suas formas de entrada e saída, criando uma estrutura que se assemelha a um hexágono onde o núcleo central se conecta ao mundo exterior por meio de "portas e adaptadores".

Na prática, Cockburn sugere que o software deve ser construído de dentro para fora. Primeiro, define-se a lógica essencial que descreve o que o sistema realmente faz, sem se preocupar ainda com como ele interage com o usuário, o banco de dados ou outros serviços. Esse núcleo interno representa o domínio puro e independente de tecnologia. Depois, criam-se "portas" — interfaces que descrevem os meios de comunicação entre esse domínio e o ambiente externo. Por fim, entram os "adaptadores", que traduzem essas interações para tecnologias específicas. Um adaptador pode ser, por exemplo, uma camada REST, um repositório que fala com o banco de dados ou uma interface gráfica que coleta dados do usuário.

Essa abordagem resolve um problema clássico do desenvolvimento tradicional: a inversão da dependência. Em sistemas construídos sem uma separação clara, a regra de negócio acaba dependendo de frameworks e bibliotecas externas, o que inverte a lógica natural da arquitetura. A proposta hexagonal é justamente o contrário: é o mundo externo que deve depender do domínio. Esse princípio antecipa o que depois seria formalizado por Robert C. Martin no conceito de *Clean Architecture* e pelo padrão *Dependency Inversion* do SOLID. Cockburn, de certa forma, é um dos precursores dessa linha de pensamento, que coloca a estabilidade e a longevidade do domínio no centro do design.

Um exemplo real ajuda a visualizar melhor. Imagine uma aplicação de e-commerce que processa pedidos. O núcleo do sistema contém as regras de negócio: calcular o valor total, aplicar descontos, verificar o estoque e emitir faturas. Essas funções não deveriam "saber" se os dados vêm de um banco MySQL, de uma API REST ou de um formulário HTML. Na arquitetura hexagonal, o módulo que contém essas regras expõe "portas" — como uma interface `PedidoRepository` para persistência ou `NotificadorCliente` para comunicações. Já os "adaptadores" concretos se encarregam de implementar essas interfaces: um adaptador pode salvar dados em um

banco SQL, outro pode usar um serviço em nuvem, e um terceiro pode apenas armazenar tudo em memória para testes. Essa separação permite, por exemplo, que um desenvolvedor troque o banco de dados por outro ou adicione uma nova forma de interação (como um chatbot) sem tocar nas regras de negócio.

Essa flexibilidade se mostra especialmente útil em contextos de evolução tecnológica rápida. É comum que um produto digital comece como um protótipo simples — talvez com persistência em arquivos locais e uma interface web rudimentar — e depois precise crescer para um ambiente escalável na nuvem, com microsserviços e bancos distribuídos. Em uma arquitetura tradicional, cada uma dessas mudanças poderia exigir uma reescrita completa. Já com a arquitetura hexagonal, basta criar novos adaptadores. O domínio permanece o mesmo. Esse princípio se alinha muito bem com o conceito de *evolvability*, ou capacidade de evolução, fundamental em empresas que trabalham com ciclos de entrega contínuos e práticas ágeis.

Outro ponto importante destacado por Cockburn é a testabilidade. Como o núcleo da aplicação é independente de infraestrutura, ele pode ser testado de forma isolada, com *mock* dos adaptadores. Isso torna possível realizar testes unitários de regras complexas sem a necessidade de configurar banco de dados, redes ou autenticação. Em um cenário real de uma fintech, por exemplo, isso permite validar o cálculo de taxas, limites de crédito e auditorias de transações de forma rápida e segura. Em termos de produtividade, essa separação também favorece equipes multidisciplinares: enquanto um time trabalha na lógica de negócios, outro pode desenvolver novos adaptadores — por exemplo, uma API pública ou um conector para sistemas externos — sem interferir na estabilidade do domínio.

Entretanto, Cockburn também reconhece que a arquitetura hexagonal não é uma “bala de prata”. Em projetos pequenos, a criação de múltiplas camadas e interfaces pode parecer burocrática e gerar um overhead inicial. Além disso, exige uma disciplina arquitetural que nem todas as equipes possuem. Implementar a separação de dependências requer experiência e uma cultura técnica madura. Em startups em estágio inicial, por exemplo, pode haver resistência em adotar essa abordagem devido à pressão por entregas rápidas. Mas, ironicamente, é nesses contextos que os benefícios da arquitetura hexagonal se tornam mais claros no médio prazo: quando o produto começa a escalar e a complexidade cresce, essa separação entre o domínio e a infraestrutura evita o temido “big ball of mud” — o sistema confuso e frágil que não suporta mudanças sem quebrar outras partes.

Em síntese, o artigo de Cockburn é mais do que uma proposta técnica; é uma filosofia de design que convida o desenvolvedor a pensar no software como uma entidade viva e sustentável. Ele nos lembra que o verdadeiro valor de um sistema está no seu domínio — naquilo que ele faz de único — e que as tecnologias são apenas ferramentas

temporárias para interagir com esse núcleo. Ao propor o uso de portas e adaptadores, Cockburn não apenas descreve um padrão arquitetural, mas estabelece uma mentalidade de independência e clareza de responsabilidades. Em um mundo onde frameworks e linguagens mudam a cada poucos anos, essa ideia de um núcleo estável e protegido é o que permite às organizações construir software duradouro. A arquitetura hexagonal, portanto, permanece atual porque captura algo essencial: a arte de isolar o que é permanente do que é passageiro — uma lição que vale tanto para o código quanto para o próprio processo de inovação.