A Hexagonal Architecture, proposta por Alistair Cockburn em seu artigo, surge como uma alternativa para resolver problemas recorrentes na organização de sistemas de software, sobretudo no que diz respeito ao forte acoplamento entre lógica de negócio e elementos externos, como interface gráfica e banco de dados. A ideia central defendida pelo autor é que a aplicação deve ser construída de forma que seu núcleo — a lógica de negócio — permaneça completamente independente de quaisquer tecnologias externas, permitindo flexibilidade, testabilidade e facilidade de manutenção. Para isso, Cockburn apresenta o conceito de ports and adapters (portas e adaptadores), que se tornou uma das referências arquiteturais mais influentes nos últimos anos.

O artigo inicia destacando dois grandes problemas encontrados em arquiteturas tradicionais. O primeiro é a mistura de regras de negócio com a interface do usuário. Quando isso ocorre, mudanças simples na interface acabam exigindo modificações nas regras centrais da aplicação, tornando o sistema frágil e de difícil evolução. O segundo problema apontado é a dependência explícita de infraestrutura, como quando a lógica de negócio acessa diretamente comandos SQL ou chamadas específicas de serviços externos. Nesse cenário, qualquer alteração no banco de dados ou no serviço utilizado compromete toda a aplicação, dificultando testes e manutenções. A arquitetura hexagonal propõe que a aplicação seja desenvolvida a partir do núcleo, estabelecendo portas (interfaces) que definem contratos de entrada e saída, sem conhecimento de como serão implementadas. Os adaptadores, por sua vez, são responsáveis por conectar essas portas ao mundo externo, seja por meio de uma interface gráfica, de um banco de dados ou de um serviço de terceiros.

A metáfora do hexágono não é literal, mas simbólica. O formato com seis lados foi escolhido para representar a ideia de múltiplos pontos de conexão, sugerindo que a aplicação pode ter diversos meios de interação com o exterior. Em cada lado do hexágono podem existir diferentes adaptadores ligados a uma mesma porta. Por exemplo, um sistema de cadastro pode ter uma porta de entrada para “criar cliente”, e essa porta pode ser utilizada tanto por um adaptador REST quanto por uma interface gráfica de desktop ou até mesmo por um script de testes automatizados. Da mesma forma, para saída, a aplicação pode definir uma porta de “persistência de cliente”, sendo que essa porta pode ter adaptadores concretos para um banco de dados relacional, um banco NoSQL ou até um arquivo em memória, usado em cenários de testes.

Cockburn também distingue os adaptadores em duas categorias: os primários (driving adapters), que iniciam uma interação com o sistema, como a interface de usuário ou um teste automatizado; e os secundários (driven adapters), que são acionados pela aplicação, como repositórios de dados ou serviços de envio de mensagens. Essa separação ajuda a organizar o fluxo das dependências e a compreender de maneira clara quem comanda cada interação. Além disso, o autor destaca a importância de se começar o desenvolvimento de forma incremental, testando o núcleo do sistema com adaptadores simulados, para depois incluir gradualmente adaptadores reais de banco de dados e interfaces de usuário.

Os benefícios dessa arquitetura são claros: maior testabilidade, já que é possível validar a lógica do domínio de forma isolada; maior flexibilidade, pois trocar tecnologias externas se torna apenas uma questão de substituir adaptadores; e maior robustez, uma vez que a lógica de negócio permanece limpa, sem ser contaminada por detalhes de infraestrutura. No

entanto, Cockburn alerta que essa abordagem exige disciplina. Definir corretamente os ports e garantir que os adaptadores não assumam lógica de negócio são tarefas que requerem cuidado e maturidade da equipe de desenvolvimento. Outro ponto delicado é decidir o nível de granularidade das portas: se forem poucas, perdem expressividade; se forem muitas, podem fragmentar demais o design.

O impacto da proposta de Cockburn vai além do artigo em si. Sua ideia de isolar a lógica de domínio inspirou arquiteturas como a Onion Architecture e a Clean Architecture, que seguem princípios semelhantes, reforçando a importância de separar regras de negócio de detalhes técnicos. A simplicidade do conceito, aliada à sua eficácia, faz com que até hoje a arquitetura hexagonal seja estudada e aplicada em sistemas de diferentes portes e complexidades, permanecendo atual mesmo em um cenário tecnológico em constante transformação.

Na prática, a arquitetura hexagonal pode ser aplicada, por exemplo, em um sistema de gestão de pedidos de e-commerce. O núcleo concentra as regras de negócio (criar pedidos, calcular preços, aplicar descontos, alterar status). As portas de entrada expõem essas funcionalidades, enquanto as portas de saída definem interações externas, como salvar pedidos, processar pagamentos e enviar notificações.

Adaptadores primários (como uma API REST ou interface gráfica) conectam usuários ao núcleo, enquanto adaptadores secundários (como banco de dados, gateway de pagamento e serviços de e-mail) atendem às necessidades externas. Durante os testes, adaptadores “mock” podem substituir os reais, permitindo validar a lógica sem infraestrutura.

O benefício é um sistema mais flexível, testável e sustentável, onde trocar banco de dados ou serviço externo não afeta o núcleo, bastando ajustar ou substituir os adaptadores.