Nome: Leandro Alencar Pereira Clemente

A arquitetura hexagonal (também chamada de Ports & Adapters) propõe que uma aplicação seja construída de modo a poder ser acionada por usuários, programas, testes automatizados ou scripts em lote, independentemente da interface externa, e que seu núcleo funcione isoladamente do banco de dados ou de dispositivos externos. A ideia central é separar o “dentro” (business logic) do “fora” (interfaces, banco, UI) por meio de **portas** e **adaptadores**: portas definem protocolos ou APIs que expressam as conversações entre o núcleo da aplicação e agentes externos, e adaptadores convertem entre essas portas e a tecnologia específica utilizada (como GUI, HTTP, banco de dados, mocks etc.). Dessa forma, a aplicação “interna” não sabe o que está do outro lado da porta — ela apenas segue o protocolo da porta.

O problema que o padrão busca resolver é a **contaminação da lógica de negócio** pela camada de interface ou pela persistência: quando a lógica acaba incorporada no código da UI ou dependente do banco, torna-se difícil testá-la isoladamente, migrar para outros modos de execução (por exemplo, executar sem interface gráfica) ou substituir a persistência. A arquitetura hexagonal resolve isso ao tratar simetricamente os lados “externos” da aplicação, quebrando a visão tradicional de camadas empilhadas: há uma assimetria entre dentro e fora, mas não entre “cima” e “baixo”. O desenho em hexágono (mais por conveniência visual do que por restrição numérica) ajuda a representar múltiplas portas saindo do núcleo e vários adaptadores conectando-se a cada porta.

Cada porta pode ter vários adaptadores, para diferentes tecnologias, como UI, testes automatizados, interfaces remotas ou mocks. Por exemplo, uma porta de entrada pode receber requisições via interface gráfica, scripts de teste ou chamadas de outra aplicação; uma porta de saída pode enviar dados a um banco real, a um mock ou a alguma interface externa. Isso torna fácil testar o núcleo da aplicação com adaptadores simulados (mocks), sem depender do banco ou de uma UI, e depois substituir esses adaptadores por versões reais no ambiente de produção, sem alterar o núcleo.

Na prática, o artigo ilustra com um exemplo simples: uma aplicação de cálculo de desconto. Há duas portas — uma para obter o valor ou taxa a partir de um repositório e outra para receber entrada do usuário. Em fases de desenvolvimento, implementa-se primeiro um mock de repositório, testa-se via um framework de teste (FIT), depois adiciona-se UI simulada ou real, e por fim integra-se com banco real, trocando os adaptadores. O núcleo da aplicação (a lógica de cálculo) permanece inalterado durante essas transições.

Cockburn distingue dois tipos de adaptadores (ou portas): **primários** (driving) — que conduzem a solicitação à aplicação, por exemplo, interfaces de usuários ou testes — e **secundários** (driven) — que representam serviços que a aplicação

invoca, como repositórios ou interfaces externas. Nos diagramas tradicionais em camadas, adaptadores primários ficam "acima" e os secundários "abaixo", mas no contexto hexagonal essa distinção é apenas uma consequência da aplicação real do padrão.

O autor também aborda como casos de uso (use cases) devem ser escritos com foco no **frontier** — ou seja, no nível da interface da aplicação (o interior da arquitetura), sem referência a tecnologias externas. Isso resulta em casos de uso mais coesos, legíveis e estáveis ao longo do tempo. A quantidade de portas não é fixa: pode-se usar poucas (duas, três, quatro) ou abstrair várias funcionalidades em portas maiores. O critério é intuitivo segundo o domínio da aplicação.

O artigo também aponta usos conhecidos: por exemplo, um sistema de alertas meteorológicos tinha quatro portas naturais (feed de alerta, interface administrativa, interface de notificação ao usuário, interface de dados de assinantes). O padrão permitiu adicionar novas interfaces (como HTTP ou e-mail) apenas criando novos adaptadores para portas existentes sem modificar o núcleo. Em outro exemplo, equipes distribuídas podem usar mocks e testes isolados via ports & adapters para desenvolver subsistemas de forma independente, integrando-os depois por meio de adaptadores.

Além disso, o artigo relaciona o padrão a conceitos clássicos: ele é uma aplicação especializada do padrão **Adapter** (do catálogo Gang of Four); também se conecta ao **Princípio da Inversão de Dependência** (Dependency Inversion), pois tanto módulos de alto nível (lógica) quanto módulos de baixo nível (persistência) dependem de abstrações (portas), e os detalhes (adaptadores) dependem dessas abstrações. Também discute padrões complementares, como mocks, loopback, pedestais e o uso de frameworks de injeção de dependência (como Spring) para instanciar adaptadores.

Em síntese, a arquitetura hexagonal propõe um modelo robusto e flexível, que ajuda a manter o núcleo de aplicação limpo, testável e indiferente às tecnologias externas. Isso favorece a evolução do software, facilita testes automatizados e incrementa a modularidade e adaptabilidade da aplicação ao longo do tempo.