O artigo “Hexagonal Architecture: The Original 2005 Article” aborda uma questão que boa parte dos desenvolvedores conhece por experiência própria: quando a lógica de negócio de um sistema fica misturada com a interface de usuário, ou com o banco de dados, o código vira difícil de testar, difícil de evoluir e acaba preso a tecnologias específicas. Cockburn parte desse gargalo para apresentar o que ele chama de “ports & adapters” ou “arquitetura hexagonal”. A ideia é simples, mas poderosa: desenhar o sistema de forma que a lógica central (o ‘coração’ do software) seja isolada do mundo exterior — seja da UI, de scripts, de bancos de dados ou de sistemas parceiros — por meio de “portas” que conectam adaptadores tecnológicos.

Ele explica que normalmente muitos sistemas sofrem com dois problemas simétricos: de um lado, a lógica de negócio se infiltra na UI, o que impede testes automáticos, mudaquando os botões ou campos mudam; de outro, a lógica fica dependente da fonte de dados ou de serviços externos, o que trava o desenvolvimento quando o banco de dados ou o serviço falha ou precisa ser trocado. O que Cockburn propõe é inverter essa dependência: o núcleo da aplicação não sabe nada sobre a UI, o banco ou o serviço externo. Ele se comunica apenas por meio de interfaces bem definidas (os “ports”), e os adaptadores (UI, scripts, banco, mocks) se ligam a essas portas.

O visual da “hexágono” aparece para representar que não se trata de uma arquitetura em camadas simples (em que “interface → lógica → banco” é fixa), mas sim de um gráfico simétrico onde o centro se comunica com vários lados — múltiplas portas — de forma igualitária. Isso permite, por exemplo, que o sistema seja dirigido por um usuário via UI, ou por um script automatizado, ou ainda por outro sistema batendo via API. Do mesmo modo, ele pode acessar um banco real, um mock ou um arquivo, sem que o núcleo da lógica se altere.

Um dos pontos mais interessantes é a ênfase em fazer com que a aplicação possa “funcionar sem UI ou banco de dados” — ou seja: testável isoladamente, em modo “headless”, com mocks. Isso facilita que se escrevam testes automatizados robustos, que o sistema não fique preso a uma configuração específica de banco de dados e que possa interagir com outros sistemas ou serviços de forma mais leve.

Cockburn também comenta sobre quantas portas são apropriadas — não há regra rígida, mas sugere que normalmente duas a quatro já são suficientes (por exemplo: usuário, banco, serviço externo, notificação). Ele mostra que escolher poucas portas torna o sistema mais manejável, embora “errar” na contagem das portas não leve a desastre, desde que se mantenha a separação núcleo-mundo externo.

Na parte de “use cases” ou casos de uso, ele alerta para o risco de escrever casos de uso muito tecnológicos, que ficam presos à UI ou aos detalhes do banco. Em vez disso,

os casos de uso devem estar no limite da aplicação — ou seja, no núcleo, onde as portas definem o que a aplicação oferece ao mundo. Essa abordagem torna os casos de uso mais curtos, mais estáveis e mais fáceis de manter.

Por fim, o que mais me marcou no artigo foi esse convite à disciplina arquitetural: separar o que muda com o que não muda, desacoplar o núcleo das variações tecnológicas. É uma perspectiva que retoma padrões de adaptação e abstração (como o Adapter Pattern, Inversão de Dependência), mas aplicada de um modo mais orgânico ao sistema inteiro. Se aplicada com consciência, a arquitetura hexagonal ajuda a manter o código limpo, fácil de testar, pronto para mudança. Se ignorada, corre-se o risco de voltar ao velho cenário de lógica misturada, difícil de evoluir.