Capítulo 7 – Arquitetura

A etapa de introdução começa com o autor nos familiarizando com o conceito de Padrões Arquiteturais e Arquitetura de Software. É importante ressaltar que existem diversas definições para ambos os tópicos, variando de autores para autores. Marco Tulio coloca diversas dessas perspectivas para o leitor, trazendo uma boa base para todo o resto do capítulo. Para mim foi possível entender o conceito de arquitetura como uma forma em alto nível de organizar o software, além de comtemplar o significado da frase de Ralph Johnson, que nos diz que arquitetura de software abrange as decisões mais importantes de um sistema. Portanto, arquitetura de software é um conjunto de decisões que uma vez tomadas, serão dificilmente revertidas no futuro. Parte dessas escolhas são a definição da linguagem de programação, o banco de dados utilizado, o padrão arquitetural, etc. Já como padrão arquitetural, compreendi o significado como uma organização dos módulos de um software e as relações entre eles. Logo, a arquitetura de software compõe uma série de decisões críticas, que englobam os padrões arquiteturais.

Em um próximo tópico, através de um debate denominado Tanenbaum-Torvalds, é possível captar uma importante mensagem, resultante de uma grande discussão: arquitetura não só engloba decisões críticas e difíceis de serem revertidas, como também são decisões que podem levar anos para expor pontos negativos e que comecem a causar problemas. Isso reforça mais ainda o fato de realizar uma análise minuciosa para que seja decidida a arquitetura de um software.

A partir deste ponto somos introduzidos a uma série de arquiteturas, começando pela Arquitetura em Camadas. É feita uma analogia muito interessante: essa arquitetura é organizada de forma que um conjunto de classes é denominado “camada”. Uma camada só se comunica com uma camada imediatamente abaixo de si, utilizando seus serviços. Assim, é estabelecida uma hierarquia como em um bolo.

Logo após, somos introduzidos a Arquitetura em Três Camadas, composta primeiramente por uma camada chamada de Interface com o Usuário, lidando desde exibição de informações até coleta de dados por meio de interface. Em seguida, Lógica de Negócio, camada que contém todas as “regras” de um negócio. Por fim, a camada de Banco de Dados, responsável por armazenar os dados obtidos pelo sistema. Na ordem explicitada, temos que a primeira camada é executada na máquina do cliente, enquanto a segunda executa em um servidor e temos por fim um banco de dados. Este seria um padrão convencional para esta arquitetura.

Continuamos para a arquitetura MVC, onde as classes são divididas em 3 módulos: Visão, onde ficam aquelas classes responsáveis pela apresentação da interface gráfica do sistema, Controladora, conjunto responsável por interpretar e tratar eventos gerados por dispositivos como o mouse e teclado, podendo chamar um método das classes modelo ou visão, e o módulo de Modelo, que armazenam os dados manipulados pelas outras camadas. Esse módulo não tem conhecimento das classes de Visão ou Controladoras. Nos é apresentada uma imagem bem interessante que resume bem essa arquitetura, onde temos uma caixa representando toda a interface gráfica, contendo os módulos de Visão e Controladores, que se comunicam entre si, e o módulo de Modelo externo à essa caixa, não se comunicando com ambos os outros dois módulos, porém o bloco da interface gráfica “observa” o Modelo. Ele ainda cita que com o surgimento da web e de frameworks como o Spring, tais frameworks foram responsáveis por expandir e adaptar o MVC para a web. Assim, os sistemas que os utilizam são “forçados” a adotar uma estrutura MVC, onde a visão é composta por páginas HTML, a controladora processa uma solicitação e gera uma nova visão enquanto a camada modelo persiste os dados em um banco de dados. Nessa nova representação web, o sistema MVC é bem semelhante ao sistema de três camadas.

No próximo tópico, chegamos a uma parte mais extensa, onde o autor aborda microsserviços. Consegui interpretar o motivo detrás da existência dessa arquitetura como um reflexo da necessidade de uma arquitetura que de fato acompanhasse metodologias ágeis. Embora o desenvolvimento de um sistema tenha sido particionado em módulos, em tempo de execução ele executará como um grande monolito. Com isso, os microsserviços surgiram: alguns grupos de módulos são independentes entre si, não compartilhando memória. Assim, o risco de uma mudança feita em um Módulo 1, por exemplo, afetar um Módulo 5, são muito mais baixas. Cada módulo ou um pequeno conjunto de módulos é então executado em processos diferentes, o que também permite que sejam utilizadas diferentes tecnologias em cada módulo, visto que são independentes entre si. O uso de microsserviços se tornou mais prático devido ao surgimento de computação em nuvem, permitindo que empresas aluguem instâncias para rodar seus serviços, incluindo mais de uma instância para o mesmo serviço, se necessário. Marco também explica que essa arquitetura é mais complexa que um monolito, visto que cada serviço é um módulo independente / com baixa dependência, ou seja, por construção são sistemas distribuídos. Assim, têm alguns pontos de atenção, como o fato de comunicações entre serviços precisarem de um protocolo de comunicação caso estejam em máquinas diferentes, algo muito comum e mais complexo que apenas chamar um método de outra classe. A latência também é um ponto de atenção, visto que a distância em que estão sendo executadas as instâncias de dois serviços que se comunicam são muito mais relevantes nesse cenário.

Enfim, o uso ou não-uso de microsserviços é um ponto muito impactante na decisão da arquitetura.

Prosseguimos para Arquitetura Orientada a Mensagens, onde basicamente clientes atuam como produtores de informações, há um serviço que organiza essas informações em fila de mensagens e os servidores são consumidores desta fila. Essa arquitetura gera dois principais benefícios, sendo desacoplamento no espaço (clientes não precisam saber quem consumirá a informação que geraram, e vice-versa para servidores) e desacoplamento no tempo (se o servidor cair, clientes podem continuar gerando mensagens / informações que serão atendidas posteriormente).

Posteriormente somos apresentados à arquitetura Publish / Subscribe, onde existem dois componentes nesta arquitetura: os assinantes e os publicadores. Assinantes assinam eventos de seu interesse, que quando publicados no serviço de Publish / Subscribe pelos publicadores, serão encaminhados àqueles que o assinaram previamente. Muito semelhante ao padrão de projeto Observer, porém esta arquitetura foi feita para sistemas distribuídos.

Próximo ao fim do capítulo, somos introduzidos a resumos de outras arquiteturas, como o modelo Pipes e Filtros, onde filtros processam dados conectados por pipes que atuam como buffers, permitindo execução paralela e flexibilidade – exemplificado nos comandos do Unix. Em seguida, aborda a arquitetura Cliente/Servidor, essencial para serviços de rede, onde clientes solicitam serviços a servidores, aplicável em sistemas de impressão, acesso a arquivos, bancos de dados e web. Além disso, são discutidas as arquiteturas peer-to-peer, em que módulos atuam tanto como clientes quanto como servidores, como observado no protocolo BitTorrent.

Encerrando o capítulo 7, somos alertados sobre um anti-padrão arquitetural. Marco ilustra o caso do “big ball of mud”, em que qualquer módulo se comunica com praticamente qualquer outro módulo, o que gera um emaranhado de dependências e um código difícil de manter. Esse problema é exemplificado com o relato de um grande sistema bancário, cujo crescimento desordenado – passando de 2,5 milhões para mais de 25 milhões de linhas de código – resultou em desafios significativos, como o aumento do tempo de aprendizado para novos engenheiros e a introdução de novos bugs com cada correção.

Com isso, a leitura deste capítulo me permitiu aprofundar meu conhecimento em arquitetura de software, dando mais embasamento para argumentar sobre as arquiteturas dos sistemas que irei desenvolver ou refatorar no mercado de trabalho. Além de entender melhor os diferentes padrões e suas aplicações, isso me ajudou a perceber a importância de escrever códigos alinhados com a arquitetura adotada pela empresa, garantindo organização, escalabilidade e manutenção eficiente, seja em um cargo como desenvolvedor ou até mesmo arquiteto de software.