Fundamentos e Princípios de Arquitetura de Software

1. Introdução à Arquitetura e Design de Software

A Arquitetura de Software funciona como a "espinha dorsal" (skeleton) de qualquer projeto; tal como o esqueleto sustenta e dá forma a um organismo, a arquitetura providencia a base necessária para que todas as funcionalidades possam coexistir de forma organizada e resiliente.

Definições Centrais

Como arquitetos, devemos ser precisos na nossa terminologia:

- **Arquitetura de Software:** é a organização fundamental de um sistema, corporizada nos seus componentes, nas relações entre si e com o ambiente, e nos princípios que regem o seu design e evolução.
- **Software Design:** É a atividade de conceção ou invenção de um esquema que transforma uma especificação de requisitos num software operacional. É a ponte que liga o que o cliente pede à codificação e testes finais.
- **Princípio:** No contexto de software, um princípio é uma lei fundamental, doutrina ou pressuposto que serve de base para diversas abordagens e conceitos técnicos.

---

2. A Hierarquia do Design e Estruturas de Sistema

Gerir a complexidade em sistemas modernos exige a organização da estrutura em diferentes níveis de abstração. Esta hierarquia permite-nos focar a atenção em áreas específicas sem perder a visão estratégica global.

Os 5 Níveis de Design

O processo de design desdobra-se em cinco níveis granulares, do macro ao micro:

1. **Sistema de Software:** A visão global e o propósito do sistema completo.
2. **Divisão em Subsistemas/Packages:** A segmentação do sistema em grandes blocos funcionais.
3. **Divisão em Classes dentro de Packages:** A organização da lógica em unidades modulares de código.
4. **Divisão em Dados e Rotinas dentro de Classes:** O detalhamento da estrutura interna das classes.
5. **Design de Rotinas Internas:** A lógica algorítmica e o fluxo de execução de cada rotina individual.

Estruturas Estáticas vs. Dinâmicas

A arquitetura manifesta-se através de duas lentes principais que definem como o sistema é construído e como ele opera:

- **Estrutura Estática:** Refere-se à organização permanente dos componentes do sistema, incluindo classes, módulos e pacotes. Esta estrutura define as relações de dependência e a hierarquia entre os elementos do sistema.

- **Estrutura Dinâmica:** Envolve o comportamento do sistema em tempo de execução, incluindo a interação entre objetos, a troca de mensagens e o fluxo de controlo. Esta estrutura é crucial para entender como o sistema responde a eventos e processa informações.

---

3. Determinantes e Influências na Escolha Arquitetural

Uma arquitetura não resulta apenas de requisitos técnicos; ela é o produto de uma matriz de influências sociais, comerciais e organizacionais. O arquiteto deve navegar por este ecossistema, equilibrando os interesses dos Stakeholders, a maturidade da Organização de Desenvolvimento, a sua própria Experiência e as limitações do Ambiente Técnico.

Facto Principal: Architecture é **ortogonal** à funcionalidade do sistema.

- **Mesma funcionalidade pode ter múltiplas arquiteturas**
- **Mesma arquitetura pode suportar funcionalidades diferentes**

Fatores que Influenciam a Arquitetura:

1. **Stakeholders** (necessidades, poder)
2. **Developing organization** (capacidades, processos)
3. **Background & experience of architects**
4. **Technical environment** (plataforma, ferramentas)

O Dilema da Funcionalidade e o Fator Risco

Um axioma fundamental: os requisitos funcionais não determinam a arquitetura. Se dois arquitetos recebessem os mesmos requisitos, produziriam soluções diferentes, pois a arquitetura foca-se em como o sistema atinge as suas qualidades.

Adotamos aqui uma abordagem "Risk-Driven" (baseada no risco). Isto significa que o esforço dedicado à arquitetura deve ser proporcional ao risco do projeto:

- **Just Enough Architecture:** Se o risco for baixo ou se estivermos a reutilizar uma arquitetura comprovada, podemos (e devemos) reduzir o trabalho arquitetural.
- **Risco Elevado:** A arquitetura é crítica quando o espaço de solução é pequeno, o risco de falha é catastrófico ou os requisitos de qualidade são extremos.

A "Grande Bola de Lama" (Big Ball of Mud)

Na ausência de uma escolha consciente, o sistema degrada-se para uma Big Ball of Mud. Diferente do "código esparguete" genérico, este conceito define especificamente a ausência de uma arquitetura percetível. Todo sistema tem uma arquitetura (documentada ou não), mas a "bola de lama" é o resultado do caos onde a gestão de dependências foi ignorada.

ABORDAGEM ORIENTADA POR RISCOS (RISK-DRIVEN ARCHITECTURE)

Axioma de Engenharia: Avoiding failure é central a toda engenharia.

Técnicas arquitecturais podem ser usadas para **mitigar risks**.

Designers descartam designs **predestinados a falhar**, preferindo aqueles com **low risk de failure**.

Building successful software = anticipating possible failures

---

4. Propriedades Estruturais: Dependências, Acoplamento e Coesão

Para criar sistemas resilientes, o arquiteto deve dominar a gestão de dependências. O objetivo é criar sistemas Laxamente Acoplados (loosely coupled), onde as alterações são localizadas e não geram efeitos dominó. Ou seja, o objetivo é **reduzir as dependências desnecessárias** entre componentes.

Conceitos de Conectividade

- **Dependências:** Definem como os componentes colaboram. **O excesso de dependências rígidas é o maior inimigo da manutenção.**
- **Coesão:** É a **medida da consistência interna de um módulo**. Um componente altamente coeso foca-se numa única responsabilidade, facilitando a compreensão e o teste.
- **Acoplamento:** Refere-se à força da dependência entre módulos. **Queremos acoplamento baixo** para permitir que partes do sistema evoluam independentemente.

Princípios Fundamentais para o Sucesso

Para evitar a degradação do sistema, aplicamos ferramentas de abstração e modularização:

- **Abstração e Encapsulamento:** Esconder a complexidade e a lógica interna.
- **Separação de Interface e Implementação:** Garante que o consumidor de um serviço não dependa da forma como ele é construído.
- **Separação de Preocupações (Separation of Concerns):** Isolar diferentes responsabilidades do sistema.
- **Decomposição e Modularização:** Dividir o problema em partes geríveis.
- **Suficiência, Completude e Primitividade:** Garantir que os módulos tenham tudo o que precisam, nada mais, e de forma fundamental.

---

5. Atributos de Qualidade e Trade-offs (Compromissos)

A arquitetura é, acima de tudo, um exercício de equilíbrio. Frequentemente, a arquitetura é ortogonal à funcionalidade: **podemos manter o que o sistema faz, mas mudar drasticamente como ele se comporta em termos de performance ou segurança**.

Um exemplo prático de impacto estratégico: uma arquitetura que suporta 100 utilizadores pode ser incapaz de escalar para 100.000 utilizadores sem uma mudança arquitetural radical, mesmo que a funcionalidade entregue ao utilizador final seja idêntica.

---

6. Modelos de Sucesso: Arquiteturas Boas, Presuntivas e de Referência

A "bondade" de uma arquitetura avalia-se pelo cumprimento das metas declaradas e não por preferências estéticas.

Regras de Ouro para uma Boa Arquitetura

- **Liderança Identificada:** Deve ser produto de um grupo pequeno com um líder claro (evitar "design por comité").
- **Priorização:** Focar numa lista hierarquizada de atributos de qualidade.
- **Documentação por Views:** Utilizar diferentes vistas para comunicar com diferentes stakeholders.
- **Implementação Incremental:** Permitir que o sistema seja construído e validado por fases.

Modelos e Padrões

Existem estruturas "esqueleto" que resolvem problemas comuns de forma eficiente:

- **Arquitetura 3-Tier:** Padrão em sistemas IT para isolar mudanças na base de dados, lógica e interface.
- **Cooperating-processes:** Ideal para sistemas operativos devido ao isolamento de falhas.
- **Arquiteturas Presuntivas:** São famílias de arquiteturas que dominam um domínio (ex: N-tier em IT). Por serem modelos de sucesso para riscos comuns, a escolha de uma alternativa requer uma justificação robusta e fundamentada.
- **Arquiteturas de Referência:** Modelos genéricos que servem de base para soluções num domínio específico.