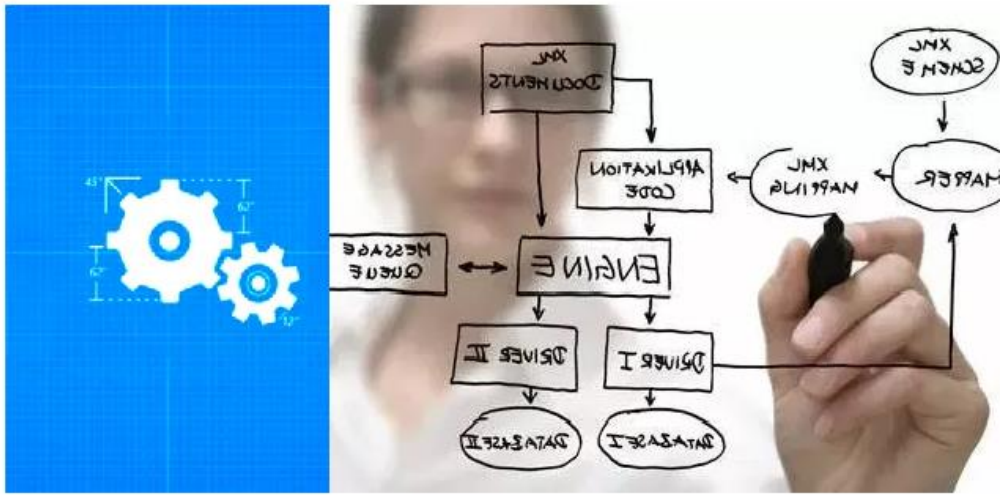
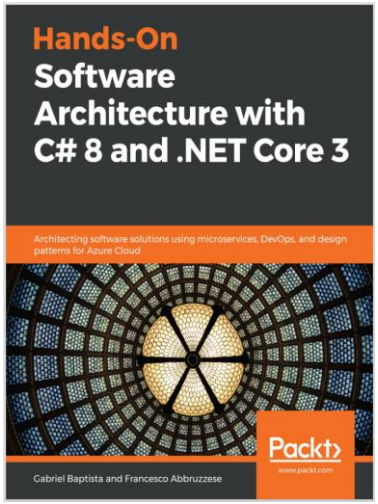
Arquitetura de Software

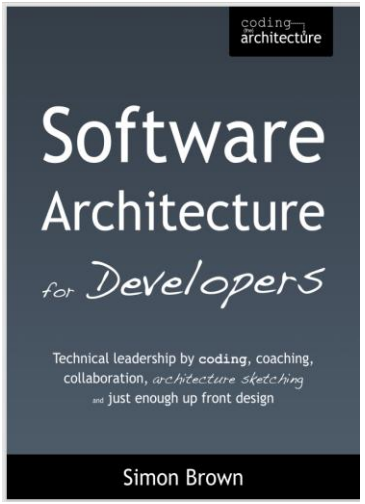
Unidade 7 - Lei de Brooks, Lei de Conway e Modelo C4

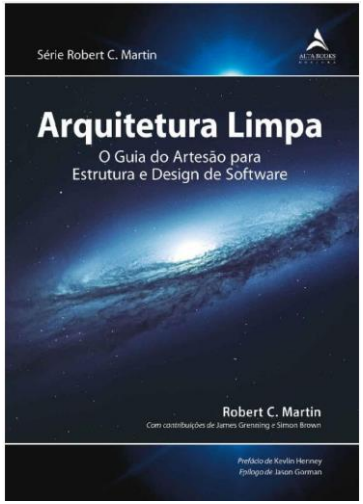


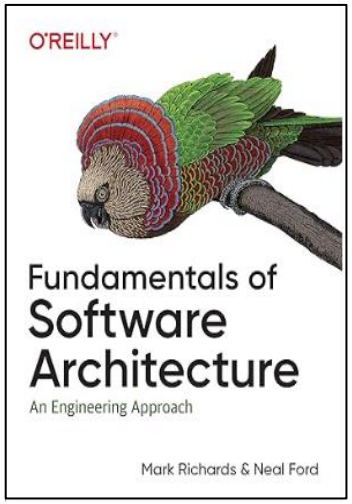
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecido.freitas@online.uscs.edu.br
aparecidovfreitas@gmail.com

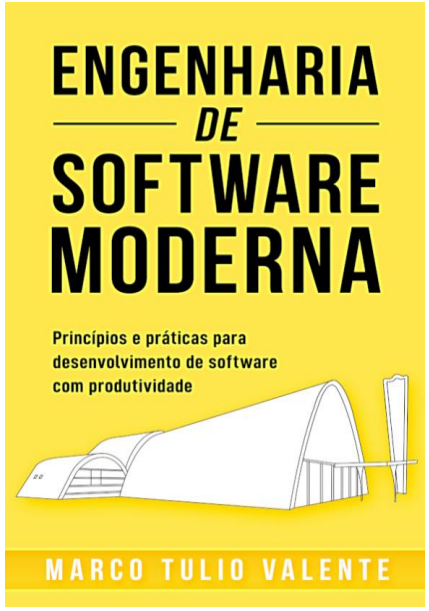
Bibliografia

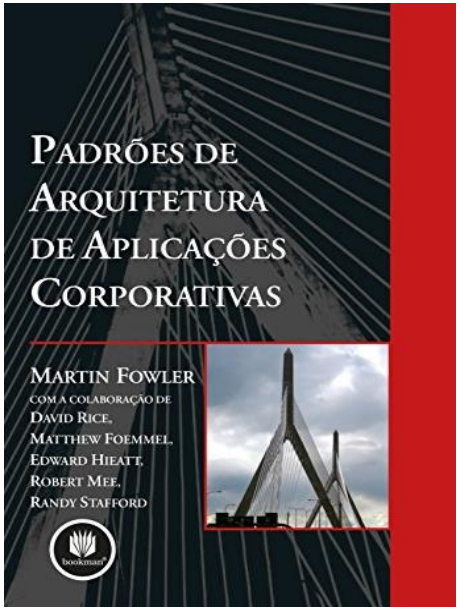












Arquiteturas de Software

- Seguem abaixo as principais arquiteturas de software:
  - Arquitetura Monolítica;
  - Arquitetura Cliente Servidor;
  - Arquitetura MVC (Model-View-Controller);
  - Arquitetura em Camadas (N-Tier);
  - Arquitetura Orientada a Serviços;
  - Arquitetura de Microsserviços;
  - Arquitetura Baseada em Componentes;
  - Arquitetura Serverless;
  - Arquitetura Orientada a Eventos.

Arquitetura Monolítica

- **Descrição**: Uma aplicação monolítica é construída como uma única unidade indivisível. Todos os componentes do software (interface do usuário, lógica de negócios, acesso a dados) estão entrelaçados e executados como um único processo.
- **Exemplo de Uso**: Aplicações de pequena escala, como um website simples ou um aplicativo de desktop para gerenciamento de inventário.
- **Histórico**: Era a norma nos primórdios do desenvolvimento de software, quando as aplicações eram menos complexas e os recursos de computação mais limitados.

Arquitetura Cliente Servidor

- **Descrição**: Separa as funções de um sistema em clientes e servidores. Os clientes solicitam serviços ou recursos, e os servidores os fornecem.
- **Exemplo de Uso**: Aplicações web onde o navegador (cliente) solicita páginas de um servidor web (servidor).
- **Histórico**: Tornou-se popular nos anos 80 e 90 com o advento das redes de computadores e da Internet.

Arquitetura MVC

- **Descrição**: Separa uma aplicação em três componentes principais: Modelo (dados), Visão (interface do usuário) e Controlador (intermediário entre Modelo e Visão).
- **Exemplo de Uso**: Amplamente usado em aplicações web para separar a lógica de negócios da interface do usuário.
- **Histórico**: Originou-se nos anos 70, mas ganhou popularidade nas últimas décadas, especialmente em frameworks web.

Arquitetura em camadas (N-Tier)

- **Foco**: Divisão do sistema em camadas físicas ou lógicas com responsabilidades distintas (apresentação, lógica de negócios, acesso a dados, persistência de dados).
- **Aplicação no Exemplo**:
  - A camada de apresentação pode ser um front-end web ou um aplicativo móvel.
  - A camada de lógica de negócios contém a lógica específica do aplicativo, como processamento de pedidos.
  - A camada de acesso a dados lida com a comunicação com o banco de dados.
  - A camada de persistência de dados envolve o armazenamento de dados em si.

Arquitetura Orientada a Serviços

- **Descrição**: Consiste em serviços autônomos que comunicam-se através de interfaces bem definidas, geralmente sobre uma rede.
- **Exemplo de Uso**: Integração de sistemas empresariais onde serviços independentes (pagamento, estoque, pedidos) são combinados para criar processos de negócios complexos.
- **Histórico**: Ganhou proeminência nos anos 2000, facilitando a integração entre sistemas de software diferentes.

Arquitetura de Microsserviços

- **Descrição**: Divide uma aplicação em um conjunto de pequenos serviços, cada um executando um processo de negócio específico e operando de forma independente.
- **Exemplo de Uso**: Plataformas como a Netflix e a Amazon, onde diferentes serviços gerenciam diferentes aspectos da aplicação (recomendações, streaming, processamento de pedidos).
- **Histórico**: Surgiu como resposta às limitações da arquitetura monolítica, ganhando popularidade na última década devido à sua escalabilidade e flexibilidade.

Arquitetura baseada em Componentes

- **Descrição**: Divide o software em componentes funcionais ou lógicos modulares e reutilizáveis.
- **Exemplo de Uso**: Aplicações onde partes específicas do software, como autenticação ou processamento de pagamento, são reutilizadas em diferentes projetos.
- **Histórico**: Tornou-se popular nos anos 90 como uma maneira de promover a reutilização de código e a modularidade.

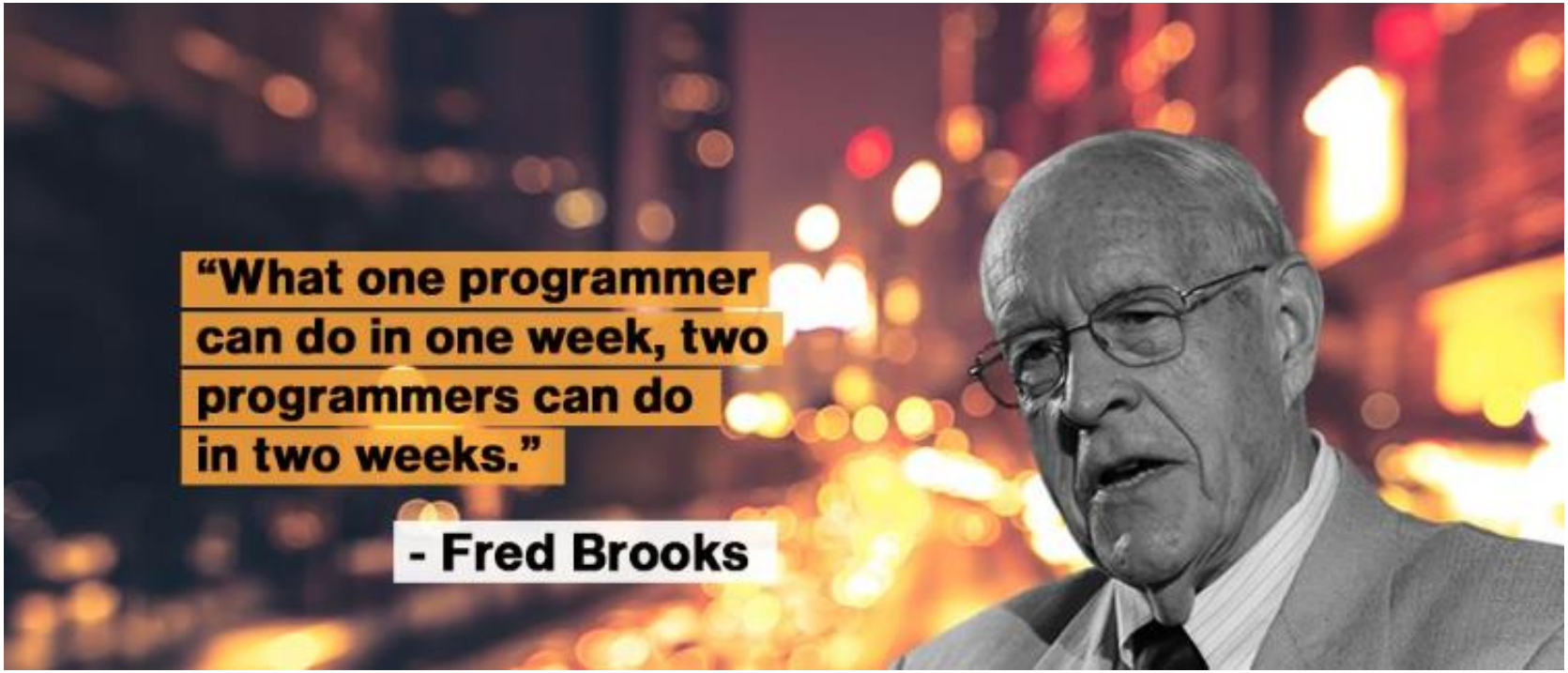
Arquitetura Serverless

- **Descrição**: Os desenvolvedores constroem e implantam funções como serviços individuais, que são executados em resposta a eventos. A gestão da infraestrutura é totalmente abstraída pelo provedor de nuvem.
- **Exemplo de Uso**: Serviços de processamento de imagem onde funções são acionadas sempre que uma nova imagem é carregada.
- **Histórico**: Emergiu com o crescimento do cloud computing, oferecendo um modelo de escalabilidade e eficiência de custos.

Arquitetura orientada a eventos

- **Descrição**: Baseia-se na produção, detecção e reação a eventos. Componentes do sistema operam em resposta a eventos significativos.
- **Exemplo de Uso**: Sistemas de negociação de ações, onde eventos de mercado acionam várias operações.
- **Histórico**: Desenvolveu-se nos anos 90 e 2000 para suportar sistemas distribuídos e reativos.

Lei de Brooks



Fred Brooks

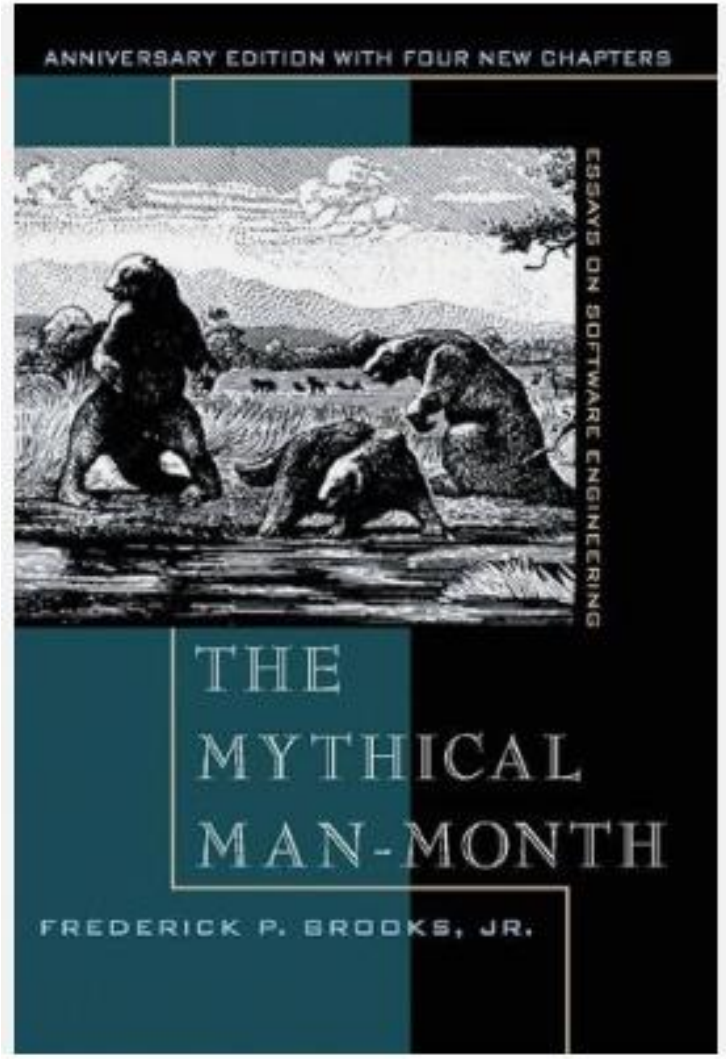
- Professor de Ciência da Computação na Universidade da **Carolina do Norte** em Chapel Hill;

- Ele é mais conhecido como o "**pai do IBM System/360**", tendo atuado como gerente de projeto para seu desenvolvimento e, posteriormente, como gerente do projeto de software "**Operating System/360**" durante sua fase de design;

- Por este trabalho, foi premiado com a Medalha Nacional de Tecnologia em 1985. Anteriormente, ele foi um dos arquitetos dos computadores **IBM Stretch** e **Harvest**.

Fred Brooks

- Autor do livro "**Mythical Man-Month**", o qual continua popular após 20 anos;
- Mais de **250.000** cópias impressas.



O PRODUTO DA PROGRAMAÇÃO DE SISTEMAS

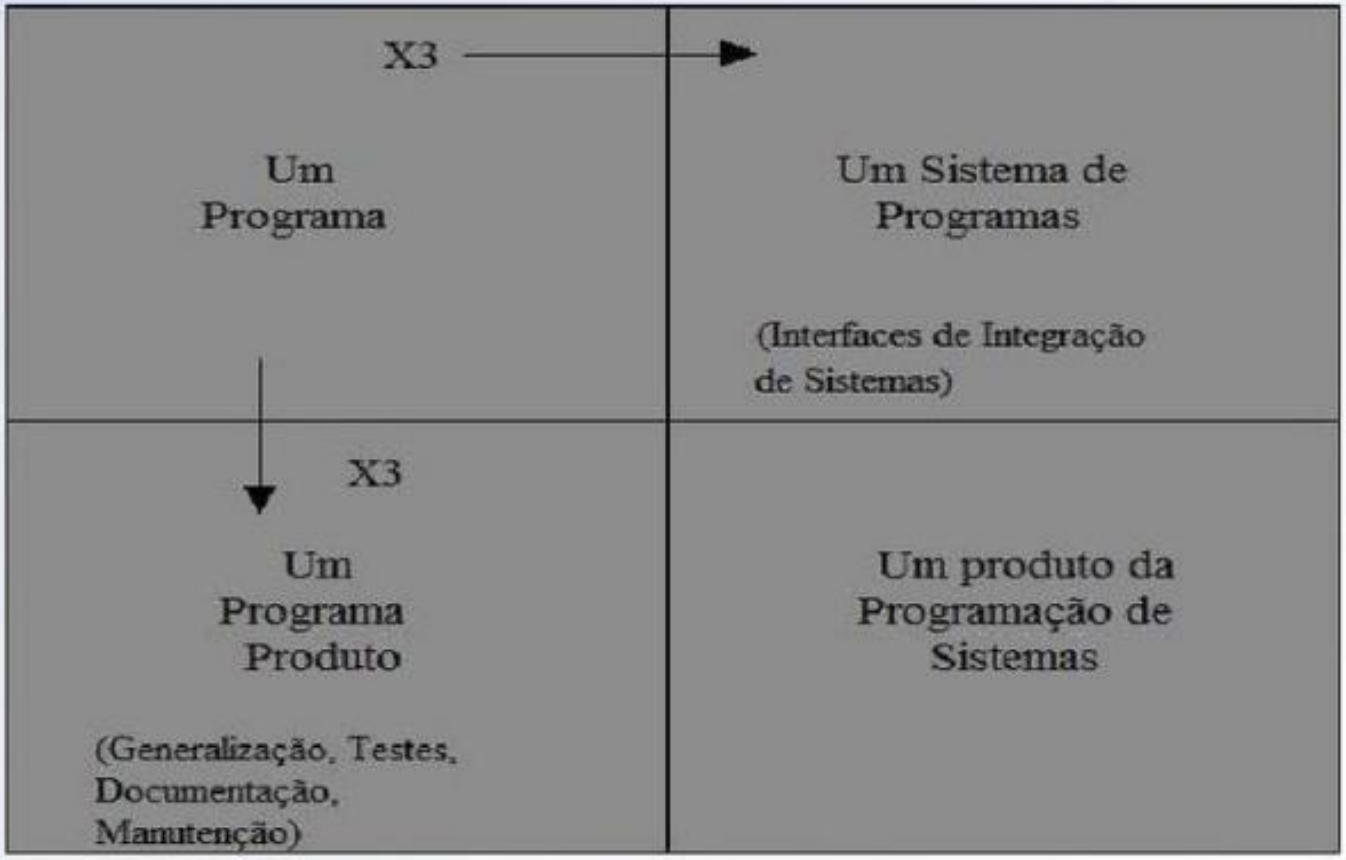


Figura 1.1 Evolução do produto da programação de sistemas

O PRODUTO DA PROGRAMAÇÃO DE SISTEMAS

- Deve ser escrito de forma generalizada;
- Deve ser EFETIVAMENTE testado;
- Deve ser CUIDADOSAMENTE documentado

O PRODUTO DA PROGRAMAÇÃO DE SISTEMAS

Programa → Componente de um sistema de programas

Valor aumenta,no mínimo, 3x

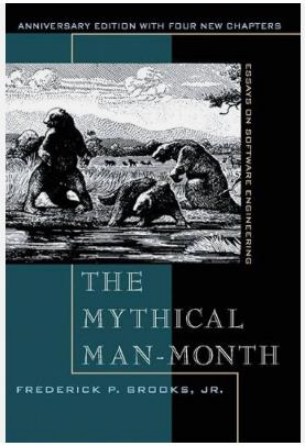
- Deve ser escrito de forma que as interfaces sejam PRECISAMENTE definidas;
- Deve ser determinado para uma quantidade DETERMINADA de recursos;
- Deve ser testado em CONJUNTO com outros componentes do sistema

O PRODUTO DA PROGRAMAÇÃO DE SISTEMAS

Programa → Produto da programação de sistemas

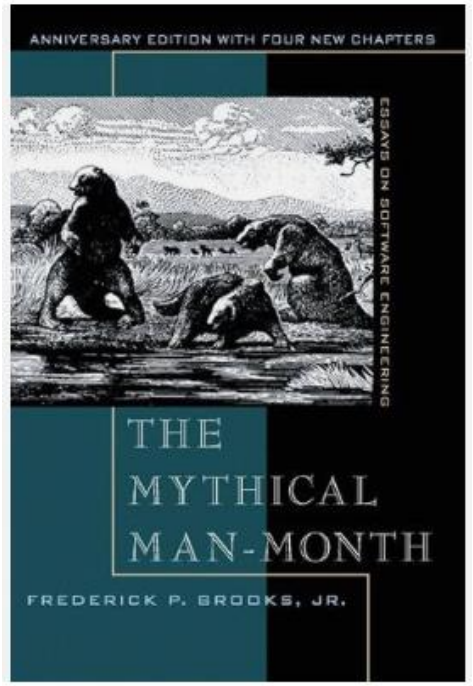
Valor aumenta 9x

- Custa **nove** vezes mais;
- Mas é o **objeto** verdadeiramente útil.



Lei de Brooks

- “Adicionar mais pessoal a um projeto de software atrasado, na verdade, o atrasará ainda mais". Isso se deve ao aumento da **complexidade** na **comunicação** e **coordenação**, bem como ao tempo necessário para treinar novos membros da equipe.

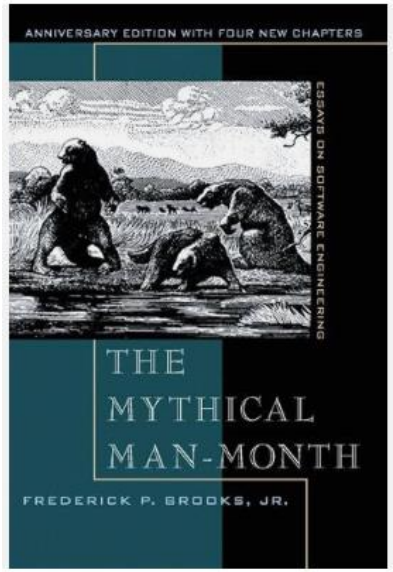


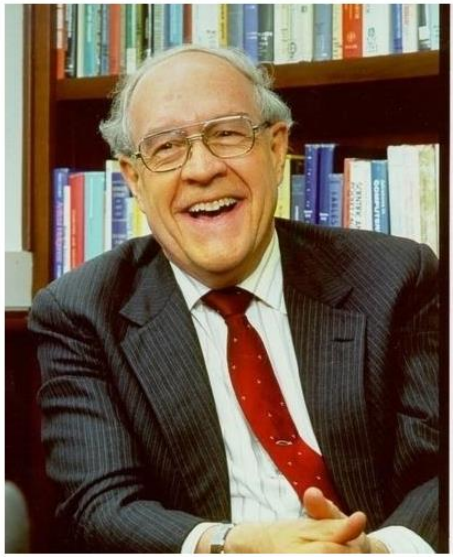
Porque a Lei de Brooks é importante para a Arquitetura de Software ?

Lei de Brooks

- Essa lei é importante para a **Arquitetura de Software** porque enfatiza a necessidade de **planejamento cuidadoso** e gestão eficaz de recursos humanos em projetos de software;

- Uma **arquitetura bem projetada** pode minimizar a complexidade e facilitar a colaboração eficiente entre os membros da equipe, reduzindo os impactos negativos de adicionar mais pessoas a um projeto em atraso.





Conway's Law

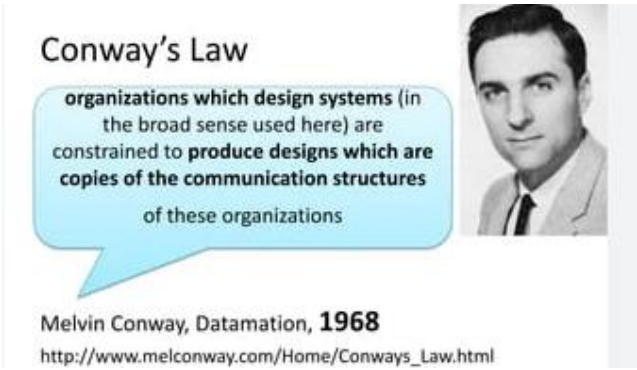
Any organization that designs a system will produce a design whose structure is a copy of the organization's internal communication structure.

Melvin Conway

- **Melvin Edward Conway** é um cientista da computação, programador que cunhou o que agora é conhecido como **Lei de Conway**;

- Desenvolveu o conceito de corrotinas. Ele cunhou o termo **coroutine** em 1958 e foi o primeiro a aplicar o conceito a um programa **assembly**;

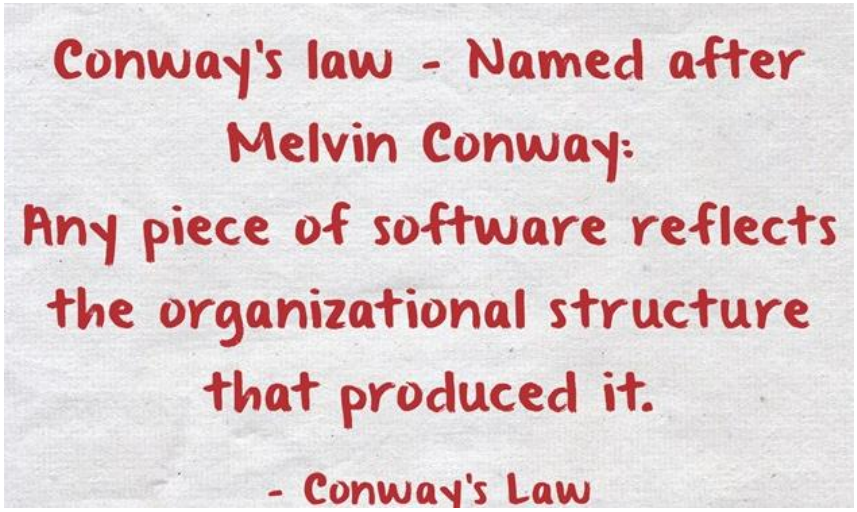
- Escreveu um assembler para o computador Burroughs modelo 220 chamado SAVE.



Lei de Conway

- A **Lei de Conway**, formulada por **Melvin Conway**, afirma que "**as organizações que projetam sistemas estão restritas a produzir designs que são cópias das estruturas de comunicação dessas organizações**“;

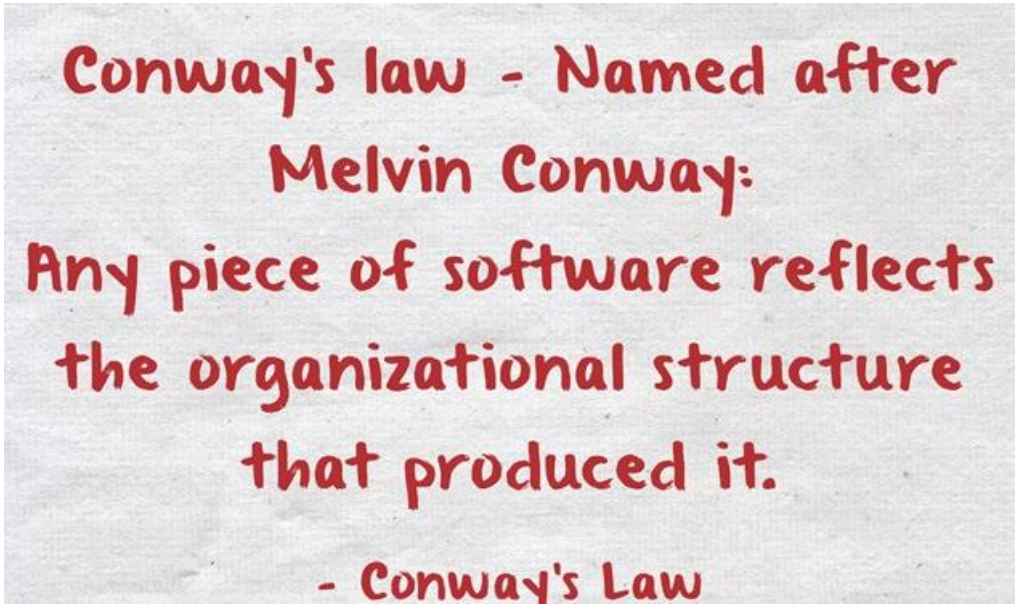
- Em outras palavras, a **arquitetura de um software** tende a refletir a estrutura organizacional da equipe que o desenvolve.



Melvin Conway

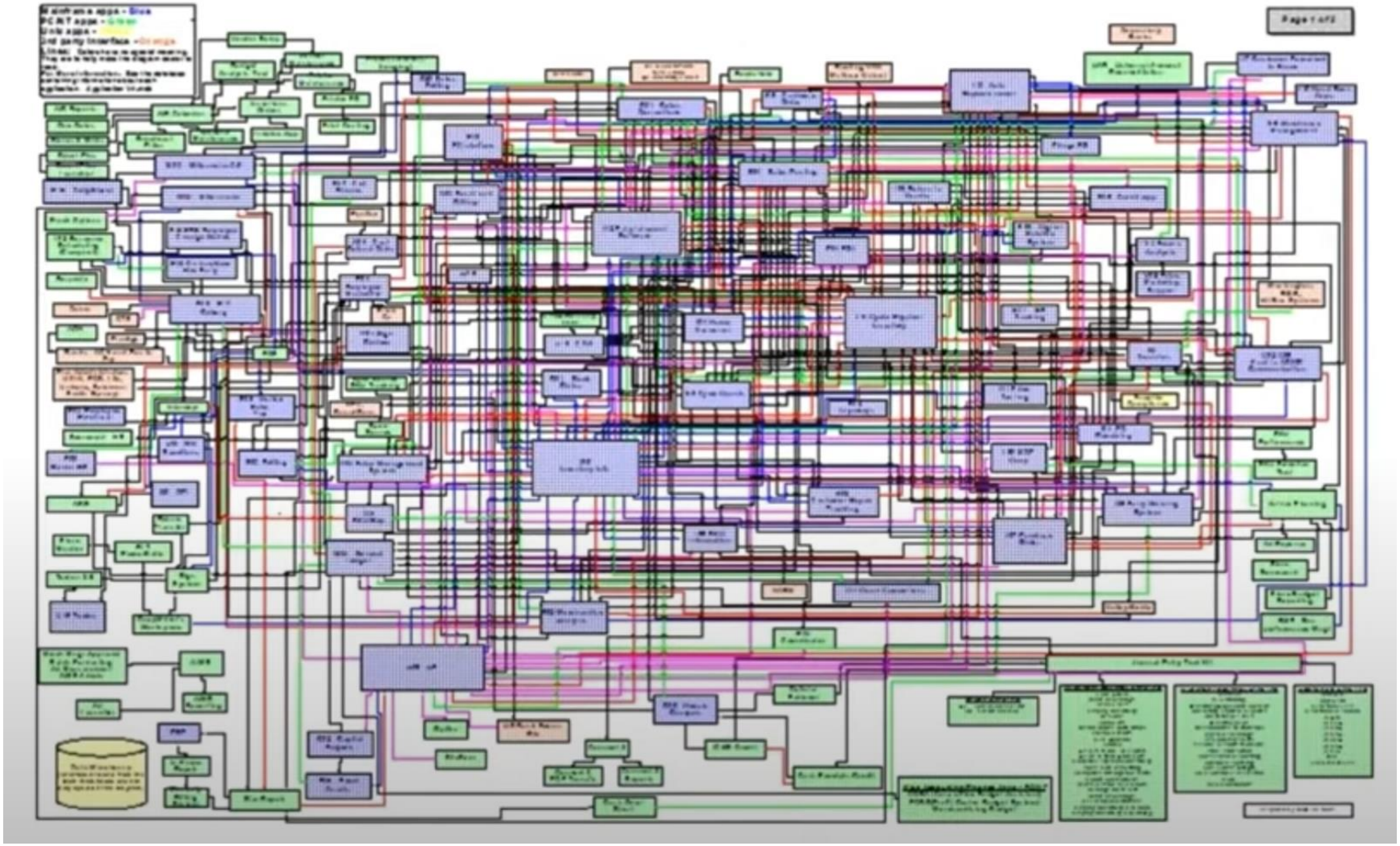
- Para a **Arquitetura de Software**, isso significa que a eficiência e a eficácia do design do software podem ser impactadas pela forma como a equipe de desenvolvimento é organizada;

- Se a equipe é fragmentada ou se a comunicação entre os membros é ineficiente, isso pode se refletir em um design de software desconexo.

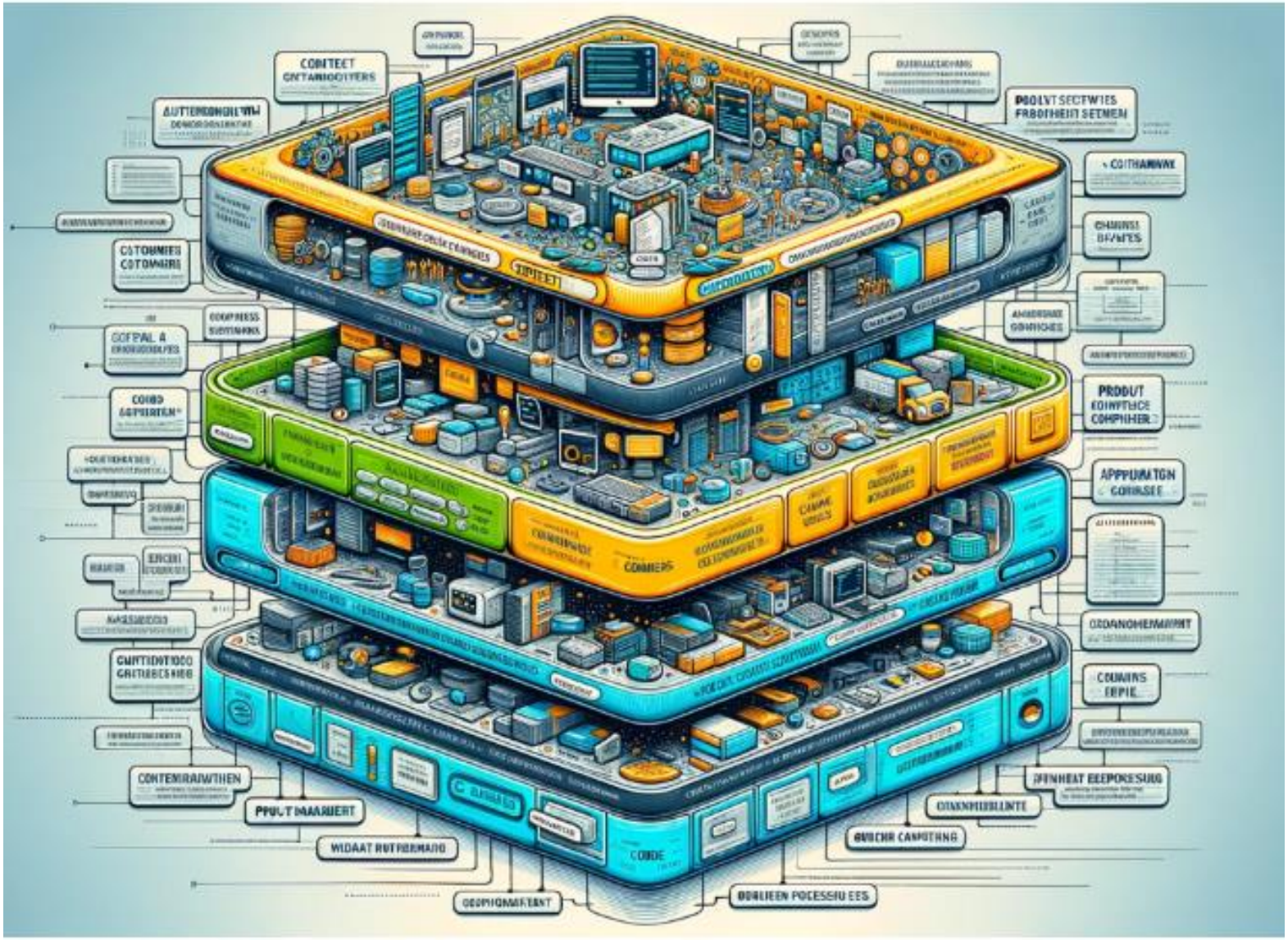


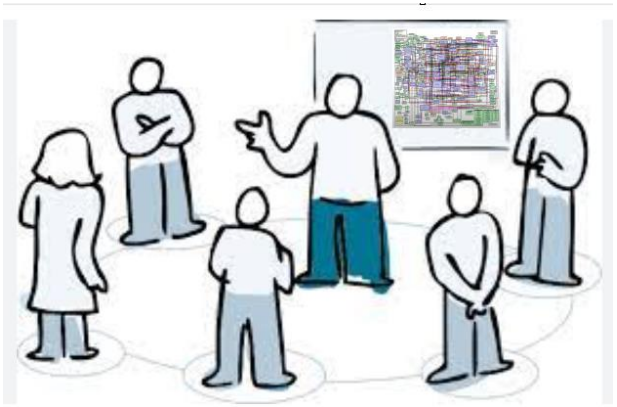
Como documentar uma Arquitetura de Software ?

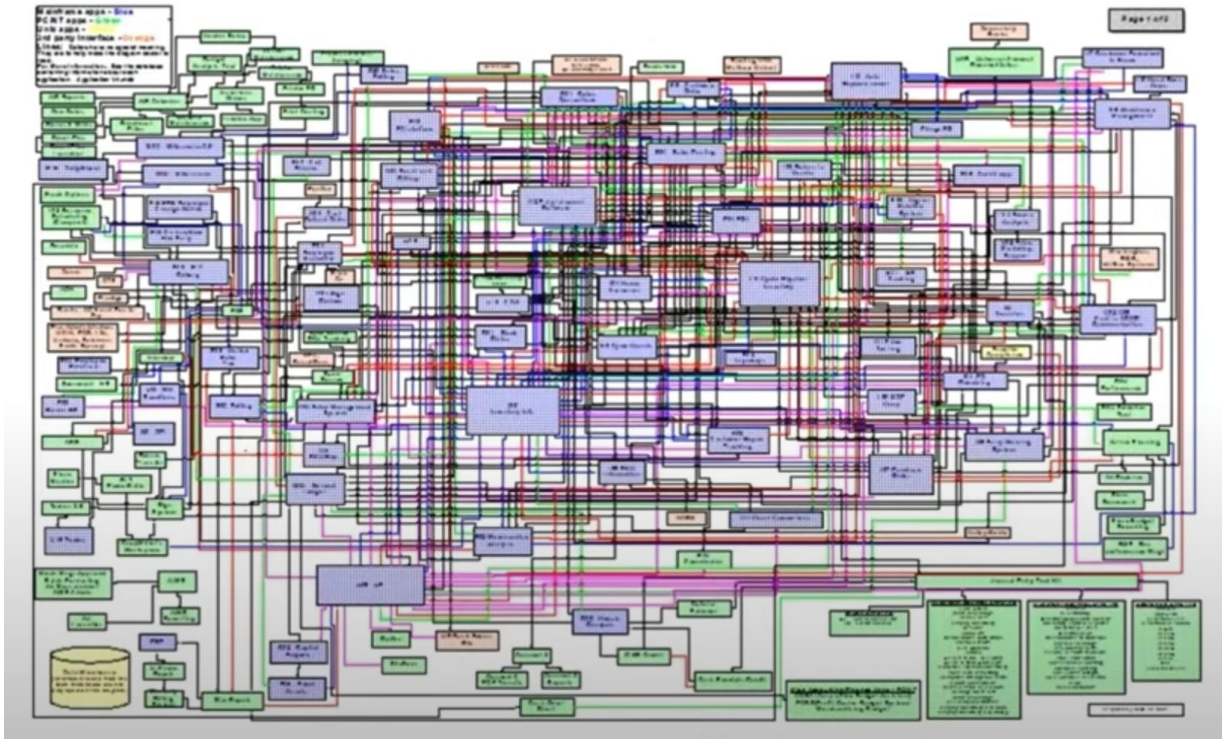
Será que este diagrama consegue definir a Arquitetura do Software ?



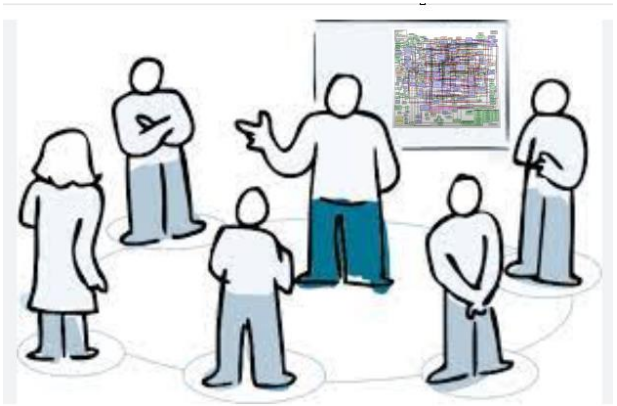
E esta ?

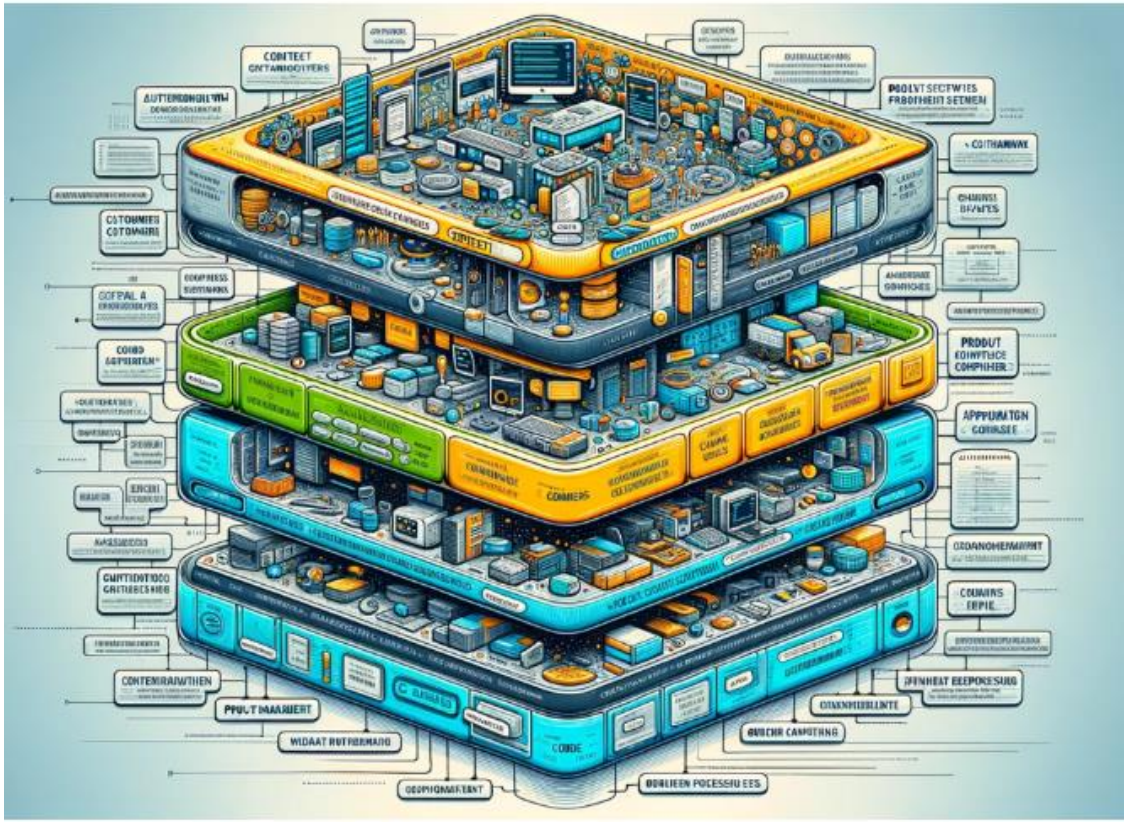






Certamente, pessoas diferentes da equipe poderão ter compreensões diferentes da Arquitetura !!!





A arquitetura do Software deve ser comunicada de forma transparente e sem ambiguidades à toda equipe de desenvolvimento !

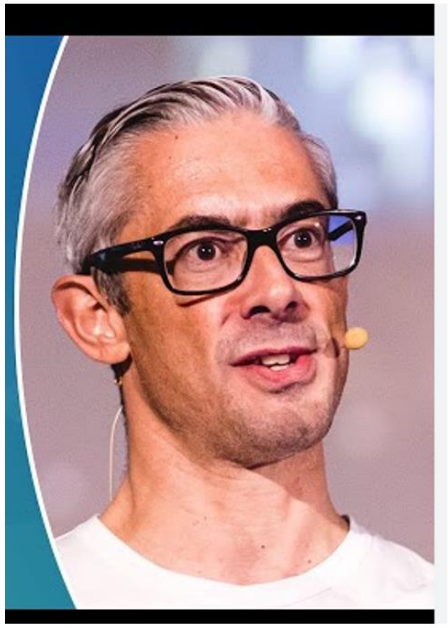
Como modelar a Arquitetura de um software para públicos diferentes?

Simon Brown


NDC {Oslo}
C4 models as code
Simon Brown
Author of "Software Architecture for Developers"

Simon Brown

- **Simon Brown** é um renomado consultor, palestrante e autor na área de arquitetura de software;
- Ele é particularmente conhecido por sua contribuição na promoção da ideia de que um arquiteto de software deve ser um parte ativa do processo de desenvolvimento de software, não apenas alguém que cria diagramas e documentos;
- Brown também desenvolveu o "**C4 Model**" para visualização de **arquitetura** de **software**



Modelo C4

c4model.com

The C4 model for visualising software architecture

Context, Containers, Components, and Code

1. System context diagram
2. Container diagram
3. Component diagram
4. Code diagram

System landscape diagram · Dynamic diagram · Deployment diagram

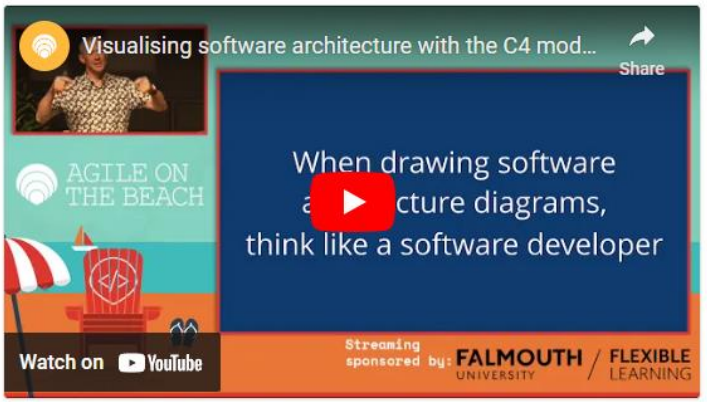
Abstractions · Notation · Tooling · FAQ · Training · Diagram review checklist · Diagram bingo

The C4 model is...

1. A set of hierarchical abstractions (software systems, containers, components, and code).
2. A set of hierarchical diagrams (system context, containers, components, and code).
3. Notation independent.
4. Tooling independent.

Uses and benefits

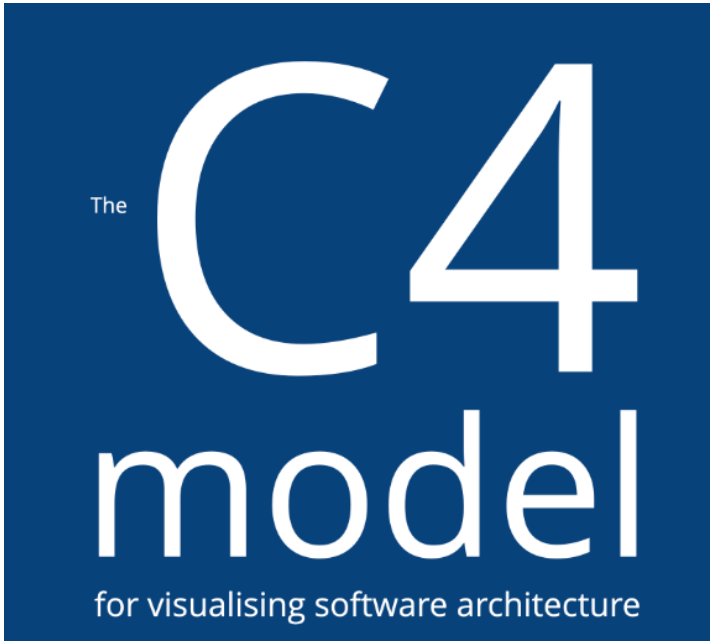
The C4 model is an easy to learn, developer friendly approach to software architecture diagramming. Good software architecture diagrams assist with communication inside/outside of software development/product teams, efficient onboarding of new staff, architecture reviews/evaluations, risk identification (e.g. risk-storming), threat modelling, etc.



Modelo C4

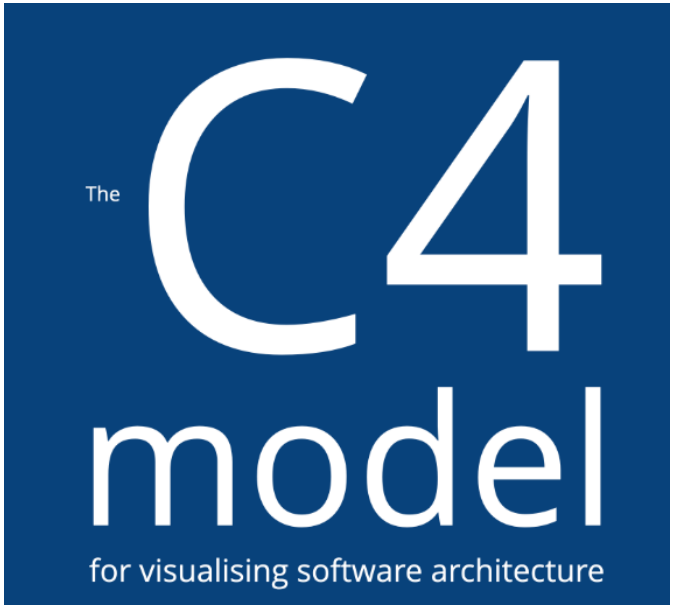
- O **modelo C4** consiste em um conjunto **hierárquico** de diagramas de arquitetura de software para contexto, containers, componentes e código;

- A hierarquia dos **diagramas C4** fornece diferentes níveis de abstração, cada um dos quais é relevante para um público diferente.



Modelo C4

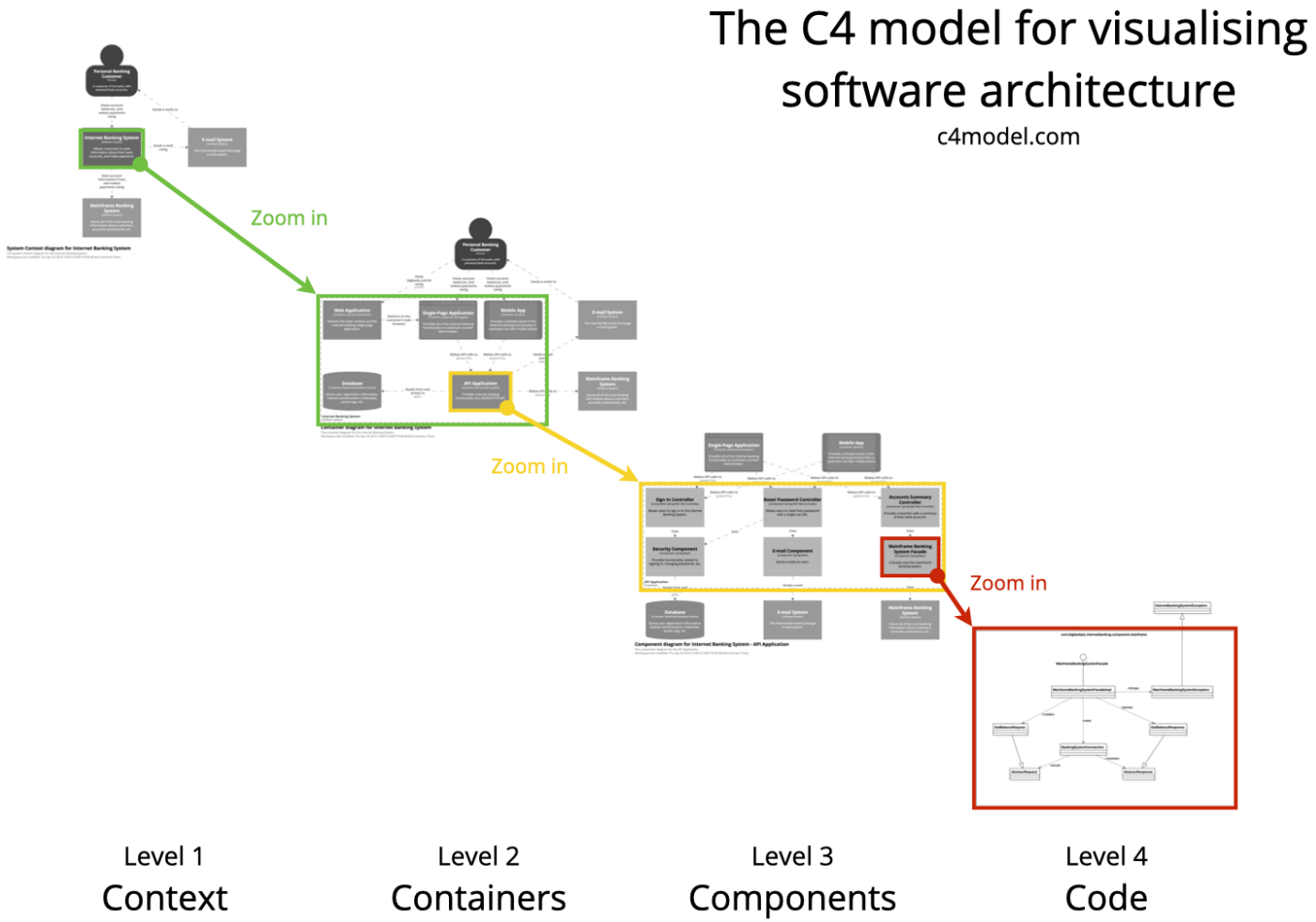
- A proposta do modelo é ser simples e comunicativa!
- O **modelo C4** apresenta quatro diagramas:
  - ✓ Diagrama de Contexto(*)
  - ✓ Diagrama de Containers(*)
  - ✓ Diagrama de Componentes
  - ✓ Diagrama de Código



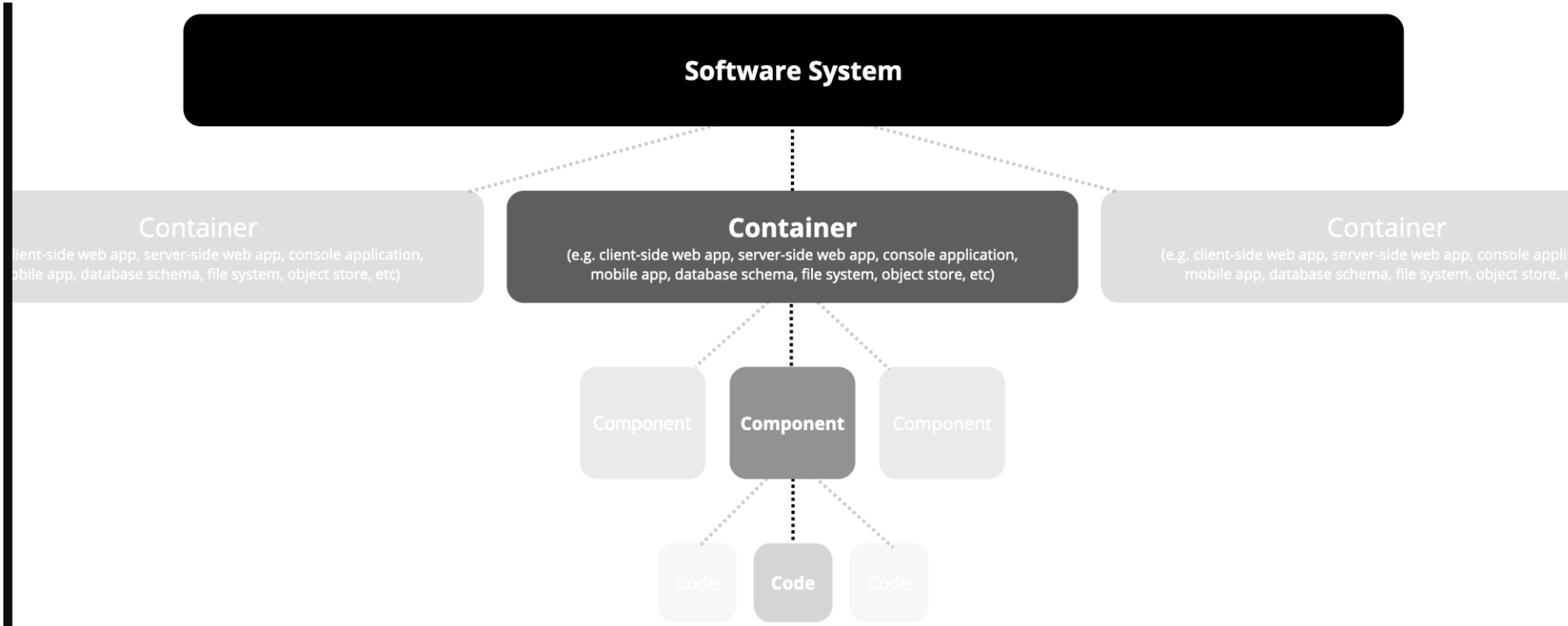
* Geralmente suficientes para se transmitir a **Arquitetura do Software**!

Modelo C4

- Os diagramas são interpretados e visualizados de forma hierárquica!



Modelo C4



A **software system** is made up of one or more **containers** (applications and data stores), each of which contains one or more **components**, which in turn are implemented by one or more **code** elements (classes, interfaces, objects, functions, etc).

Modelo C4

- Os **diagramas** são baseados em "**abstrações**" que refletem como **arquitetos de software** e **desenvolvedores** pensam e constroem software;

- O pequeno conjunto de abstrações e tipos de diagramas tornam o **modelo C4** fácil de aprender e usar;

- Observe que **não** é necessário usar-se todos os 4 níveis de diagrama; apenas aqueles que agregam valor - os diagramas de **Contexto** do Sistema e de **Container** são suficientes para muitas equipes de desenvolvimento de software.

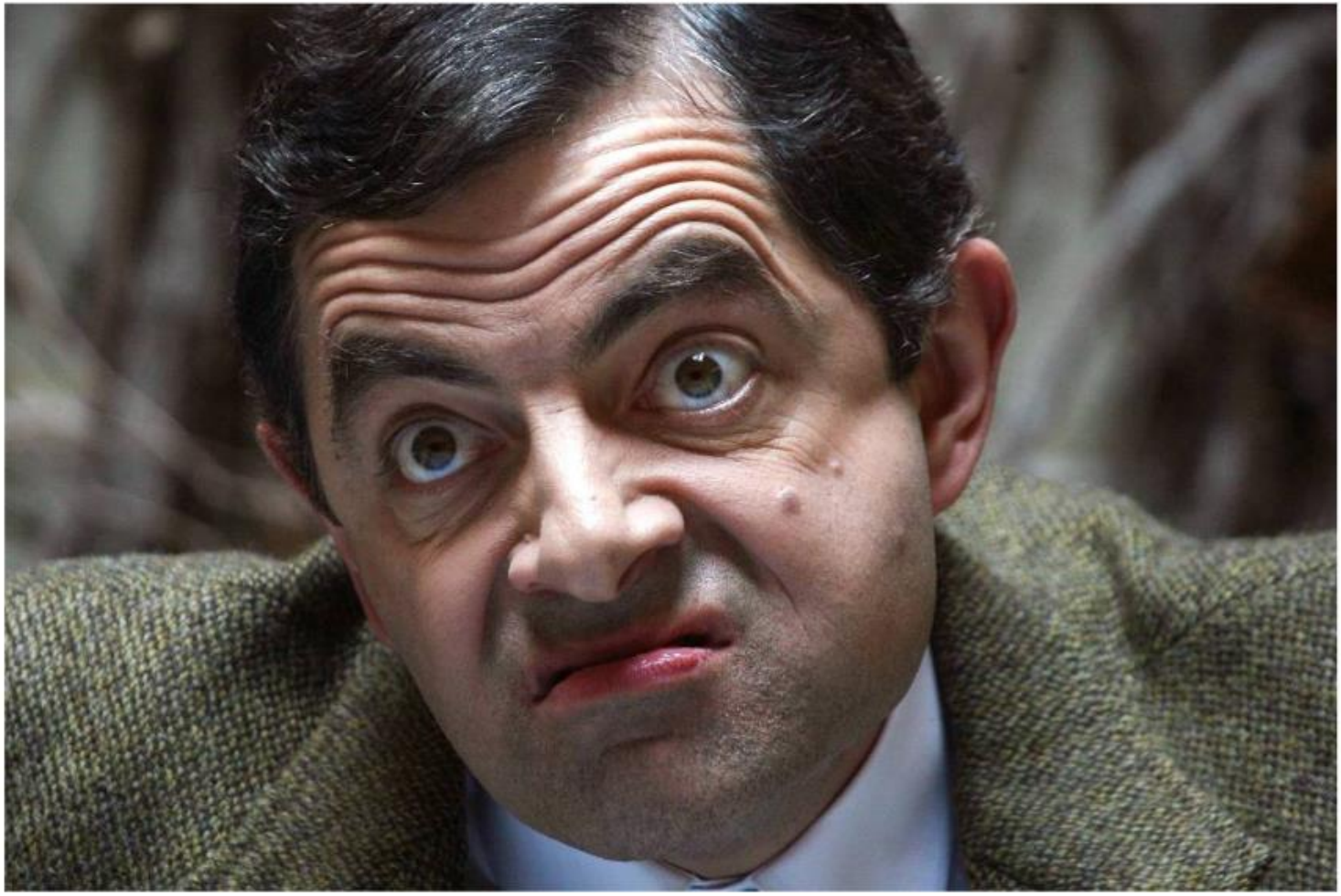
Modelo C4 - Abstrações

- Para se modelar a arquitetura do software, primeiro precisa-se de um conjunto comum de **abstrações** para criar uma linguagem **ubíqua** que possa ser usada para descrever a estrutura estática de um sistema de software;

- Um sistema de **software** é composto por um ou mais containers (aplicações e repositórios de dados), cada um dos quais contém um ou mais componentes, que por sua vez são implementados por um ou mais elementos de código (classes, interfaces, objetos, funções, etc);

- E as pessoas podem usar os sistemas de software que construímos.

Não entendi! O que significa ubíquo ???



Ubíquo

- "**Ubíqua**" é um adjetivo derivado do termo latino "**ubique**", que significa "**em toda parte**" ou "**em todos os lugares**“;

- No **contexto** em que foi usado, a expressão "**linguagem ubíqua**" se refere a uma linguagem ou terminologia que é amplamente reconhecida e entendida **no desenvolvimento e na arquitetura de software**;

- O **objetivo** de uma linguagem **ubíqua** é garantir que todos os envolvidos, independentemente de seu papel específico ou nível de conhecimento técnico, possam se **comunicar** eficientemente e entender os conceitos e componentes do sistema de software.

Modelo C4 – Abstrações

- **Pessoa**

- No **modelo C4**, a abstração "**Person**" representa um **ator humano** ou um usuário que interage com o sistema de software;

- O conceito de "**Person**" é usado para identificar e descrever as várias partes interessadas que têm alguma forma de interação ou interesse no sistema. Isso pode incluir, mas não se limita a, usuários finais, administradores de sistema, gerentes, equipes de suporte e qualquer outro tipo de usuário que possa interagir com o sistema;

- A inclusão de "**Person**" como uma **abstração-chave** destaca a importância da perspectiva do usuário e do stakeholder na arquitetura do software.

Modelo C4 - Abstrações

- **Sistema de Software**

- No **modelo C4**, um sistema de software é o nível mais alto de abstração e descreve algo que entrega **valor** aos seus usuários, sejam eles humanos ou **não**;

- Isso inclui o sistema de software que se está modelando, e **outros** sistemas de software dos quais seu sistema **depende** (ou vice-versa).

Modelo C4 – Abstrações

Containers

- No **modelo C4**, container **NÃO** é Docker !!!!
- Um container representa uma aplicação ou dados armazenados;
- Seguem exemplos de Containers no **Modelo C4**:

✓ **Server-side web application**: A **Java EE** web application running on **Apache Tomcat**, an ASP.NET MVC application running on Microsoft IIS, a **Ruby on Rails application** running on WEBrick, a **Node.js** application, etc;

✓ **Client-side web application**: A **JavaScript** application running in a web browser using Angular, Backbone.JS, jQuery, etc;

✓ **Client-side desktop application**: A Windows desktop application written using WPF, an OS X desktop application written using Objective-C, a cross-platform desktop application written using JavaFX, etc;

Modelo C4 – Abstrações

- **Containers**

- No **modelo C4**, container **NÃO** é Docker !!!!
- Um container representa uma aplicação ou dados armazenados;
- Seguem exemplos de Containers no **Modelo C4**:

  - ✓ **Mobile app**: An Apple iOS app, an Android app, a Microsoft Windows Phone app, etc;
  - ✓ **Server-side** console application: A standalone (e.g. "public static void main") application, **a batch process**, etc;
  - ✓ **Serverless** function: A single serverless function (e.g. Amazon Lambda, Azure Function, etc).

Modelo C4 - Abstrações

Componentes

- A palavra "componente" é um termo extremamente sobrecarregado na indústria de desenvolvimento de software, mas neste contexto, um componente é um **agrupamento de funcionalidades** relacionadas encapsuladas atrás de uma interface bem definida;

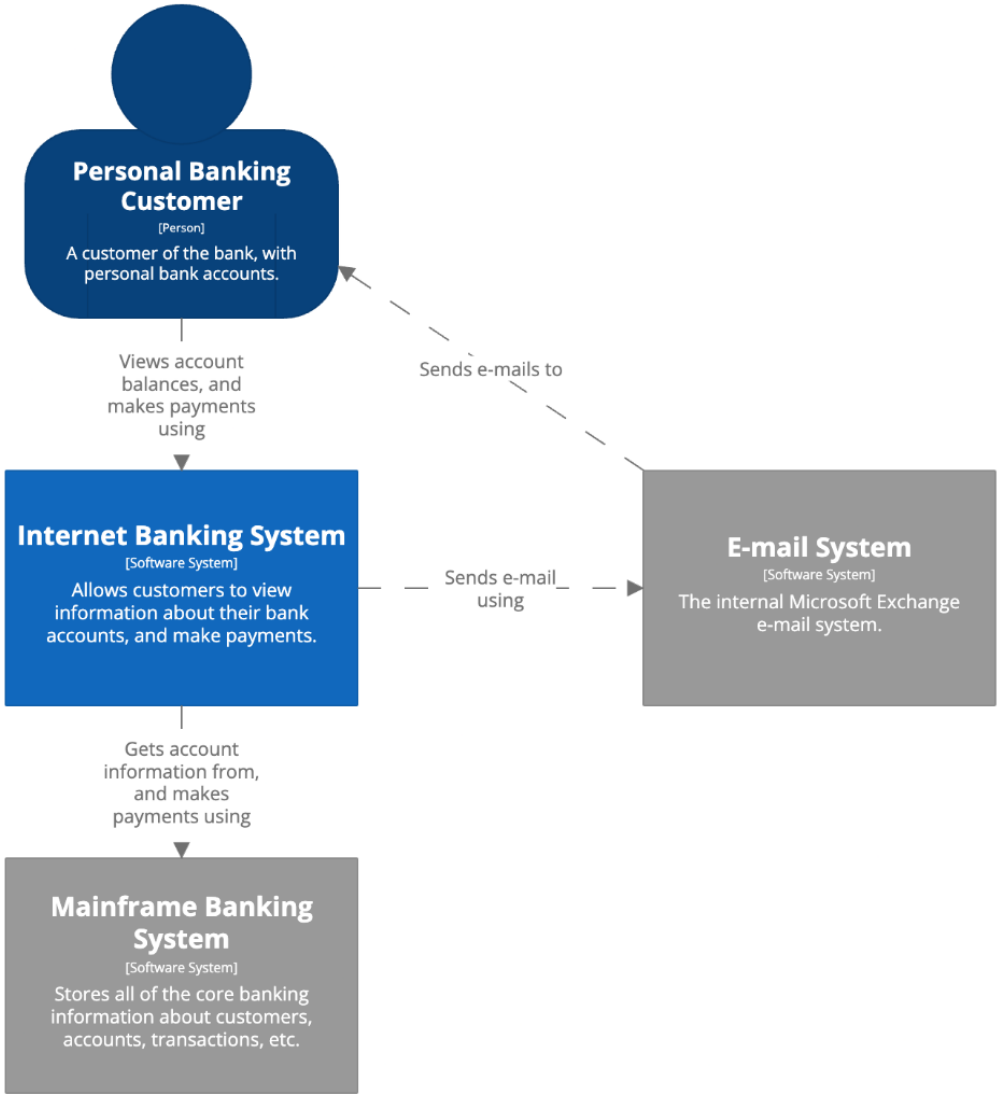
- Se você está usando uma linguagem como **Java** ou **C#**, a maneira mais simples de pensar em um componente é que ele é uma coleção de **classes de implementação** por trás de uma interface;

- Por exemplo, o **gerenciamento de LOGIN** em uma aplicação pode ser considerado um componente do sistema.

Modelo C4 – Diagrama de Contexto

- Este é o **nível mais alto** e aborda a questão: "**Por que o sistema existe**?";
- Ele fornece uma visão geral de **alto nível** do sistema, incluindo suas principais **funcionalidades**, como ele se encaixa no ambiente mais amplo, quem são os usuários principais e como eles interagem com o sistema;
- O Diagrama de Contexto inclui sistemas externos, pessoas e os principais relacionamentos entre eles e o sistema em questão.

Modelo C4 – Diagrama de Contexto



Modelo C4 – Diagrama de Container

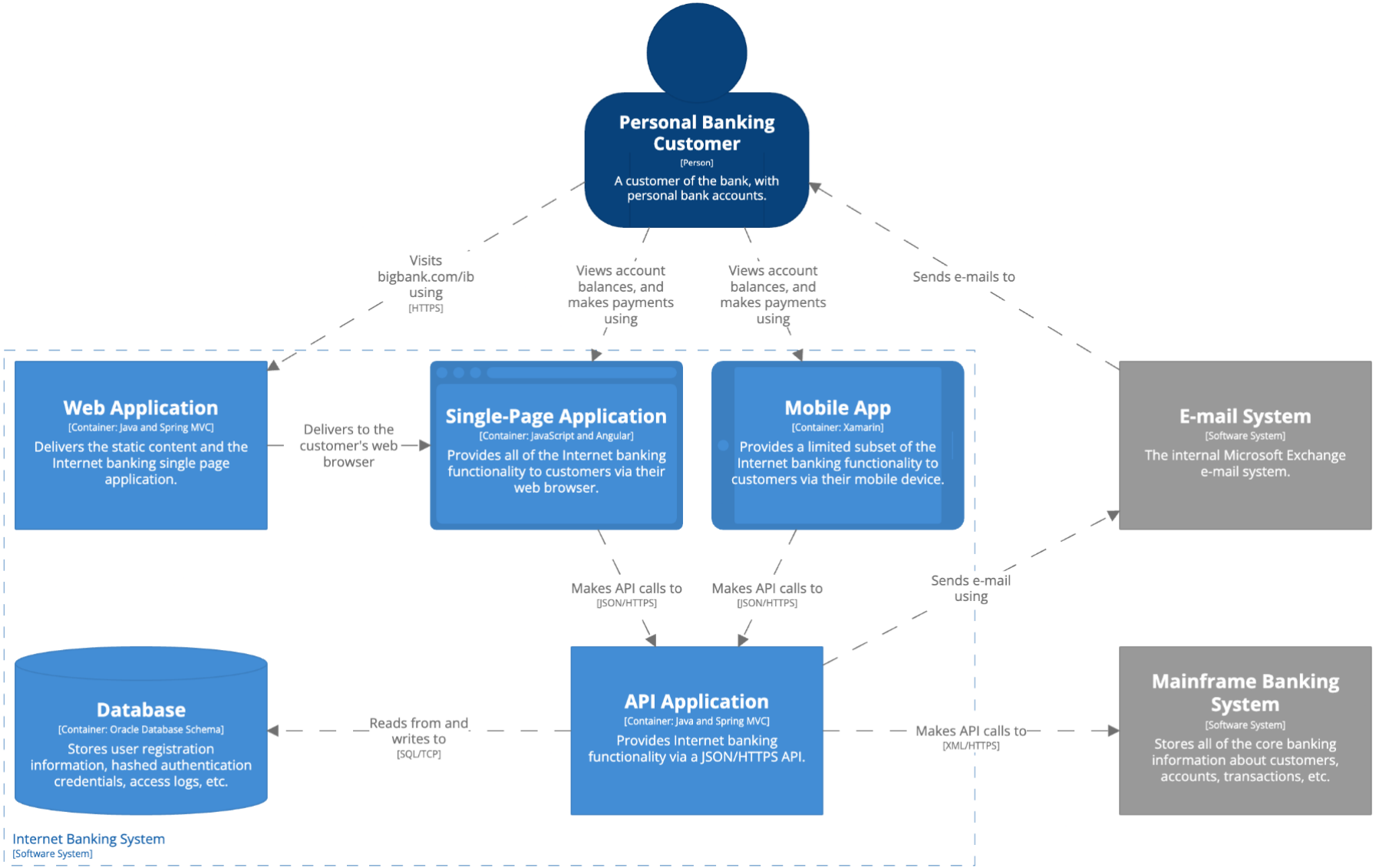
- Um "**container**" é algo como uma aplicação web do lado do servidor, uma aplicação de página única, uma aplicação de desktop, um app móvel, um esquema de banco de dados, um sistema de arquivos, etc.

- Essencialmente, um **container** é uma unidade separadamente executável/implantável (por exemplo, um espaço de processo separado) que executa **código** ou armazena dados.

- O Ddiagrama de Container mostra a forma de alto nível da arquitetura de software e como as responsabilidades são distribuídas por ela;

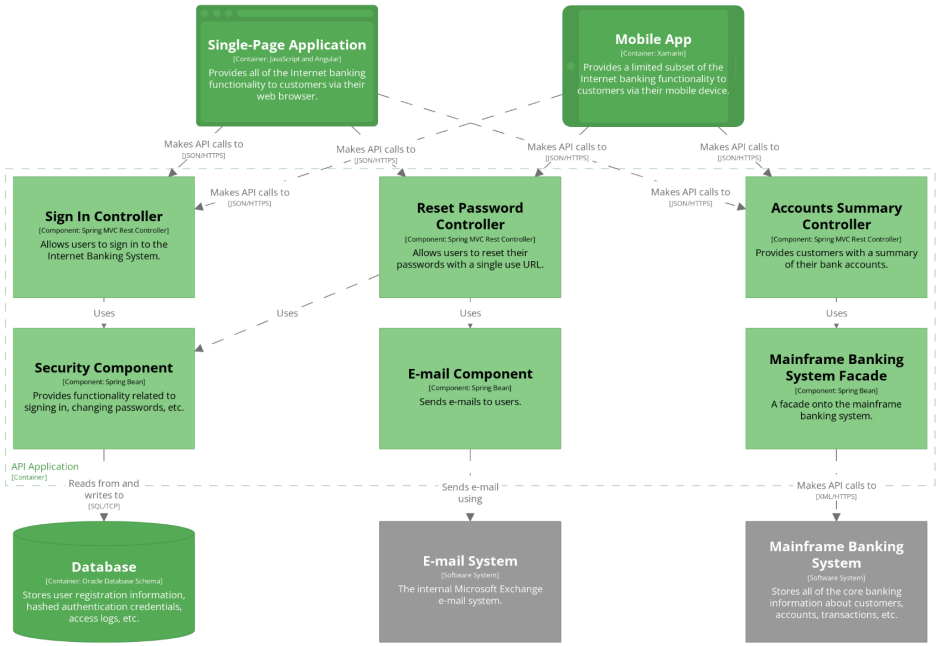
- Também mostra as principais escolhas tecnológicas e como os containers se comunicam entre si. É um diagrama simples, focado em tecnologia de alto nível que é útil tanto para desenvolvedores de software quanto para equipes de suporte/operações.

Modelo C4 – Diagrama de Container

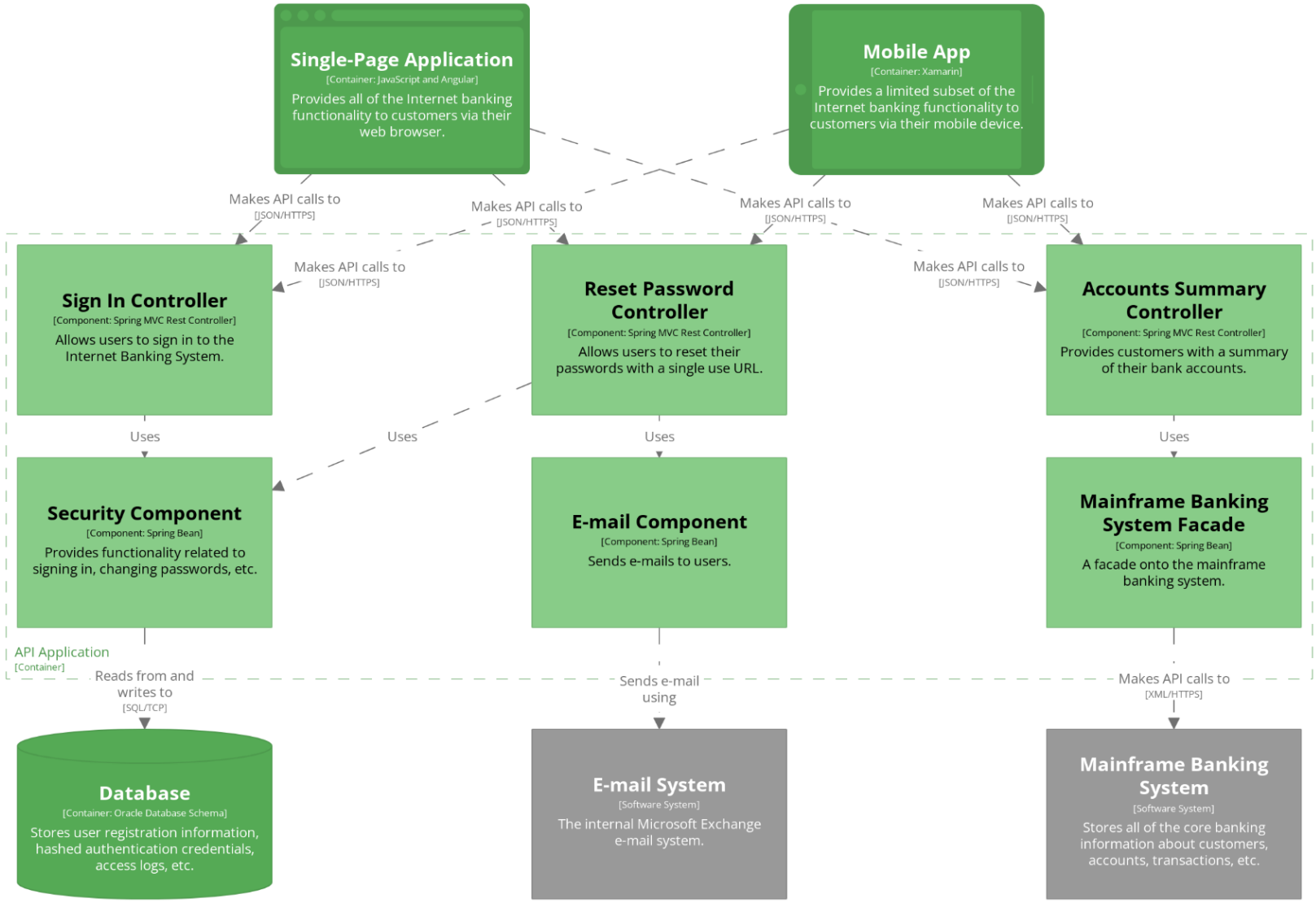


Modelo C4 – Diagrama de Componentes

- O Diagrama de Componentes mostra como o container é construído por meio de um conjunto de componentes, com suas responsabilidades e os detalhes de tecnologia/implementação.



Modelo C4 – Diagrama de Componentes



Modelo C4 – Diagrama de Código

- Cada componente pode ser expandido para mostrar a forma como o código é implementado. Usa-se **UML**, **MER** ou **similar**.