Fundamentos de Arquitetura de Software

Bootcamp: Arquiteto de Software

Junilson Pereira Souza

2020

Fundamentos de Arquitetura de Software

Bootcamp Arquiteto de Software

Junilson Pereira Souza

Sumário

Capítulo 1. Visão Geral da Arquitetura de Software

No contexto atual das organizações, é cada vez mais ubíqua a presença da tecnologia como um dos elementos primordiais para os processos empresariais, para os mais diversos fins, desde a gestão da rotina até a inovação.

Um tipo de organização que ilustra bem a importância das tecnologias são as organizações exponenciais:

> Uma organização exponencial é aquela cujo impacto ou resultado é desproporcionalmente grande - pelo menos dez vezes maior - comparado ao de seus pares, devido ao uso de novas técnicas organizacionais que alavancam as tecnologias aceleradas. (MALONE, ISMAL, GEEST).

Um aspecto central dessas organizações é o fato delas serem construídas com base nas tecnologias da informação. Uma organização exponencial tem níveis de crescimento muito maiores do que as empresas de perfil apenas linear, ou até mesmo aquelas disruptivas.

Independente da perspectiva organizacional (se a empresa foca apenas em sobrevida, se é disruptiva ou exponencial) as soluções tecnológicas têm sem mostrado algo cada vez mais presente. Como exigência para se conseguir jogar esse jogo, é necessário que as soluções tecnológicas provenham as repostas esperadas pelas necessidades de negócio.

É nesse contexto que se torna cada vez mais importante prover soluções com cada vez mais capacidade de atendimento às demandas de negócio, respeitando os critérios de qualidade técnica. Essa é uma das principais missões da arquitetura de software.

Este primeiro capítulo tem por objetivo prover uma visão geral do que é arquitetura de software, as principais dimensões tratadas nesse assunto, além de destacar quatro áreas com as quais a arquitetura faz interface e que retroalimentam

a visão arquitetural de uma solução: processo de desenvolvimento, práticas de engenharia, Devops e dados.

Definição e Elementos da Arquitetura

Arquitetura é uma daquelas palavras que soam impressionantes, usadas principalmente para indicar que se está falando de algo importante. Conforme destaca Martin Fowler, um dos mais respeitados consultores de tecnologia: “Arquitetura diz respeito a coisas importantes, seja lá o que importante signifique” (FOWLER).

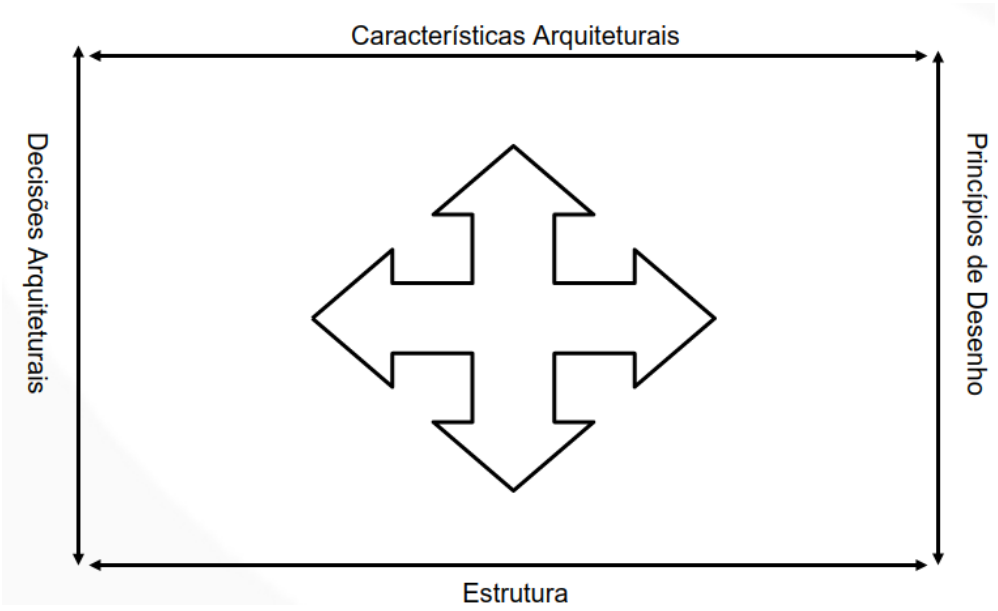
Ainda conforme o autor, o termo arquitetura envolve a noção dos principais elementos do sistema, as peças que são difíceis de mudar. Uma fundação na qual o resto precisa ser construído. Trata ainda da decomposição em alto nível de um sistema em suas partes.

Uma outra definição bastante aceita foi elaborada por autores ligados ao *Software Engineering Institute* (SEI): “Arquitetura é a estrutura dos componentes de um sistema, seus inter-relacionamentos, princípios e diretrizes que guiam o desenho e evolução ao longo do tempo” (BASS, CLEMENTS, KAZMAN).

Por outro lado, arquitetura pode ser compreendida pela coordenação de quatro dimensões (RICHARDS, FORD):

- Características arquiteturais;
- Estrutura;
- Decisões arquiteturais;
- Princípios de desenho.

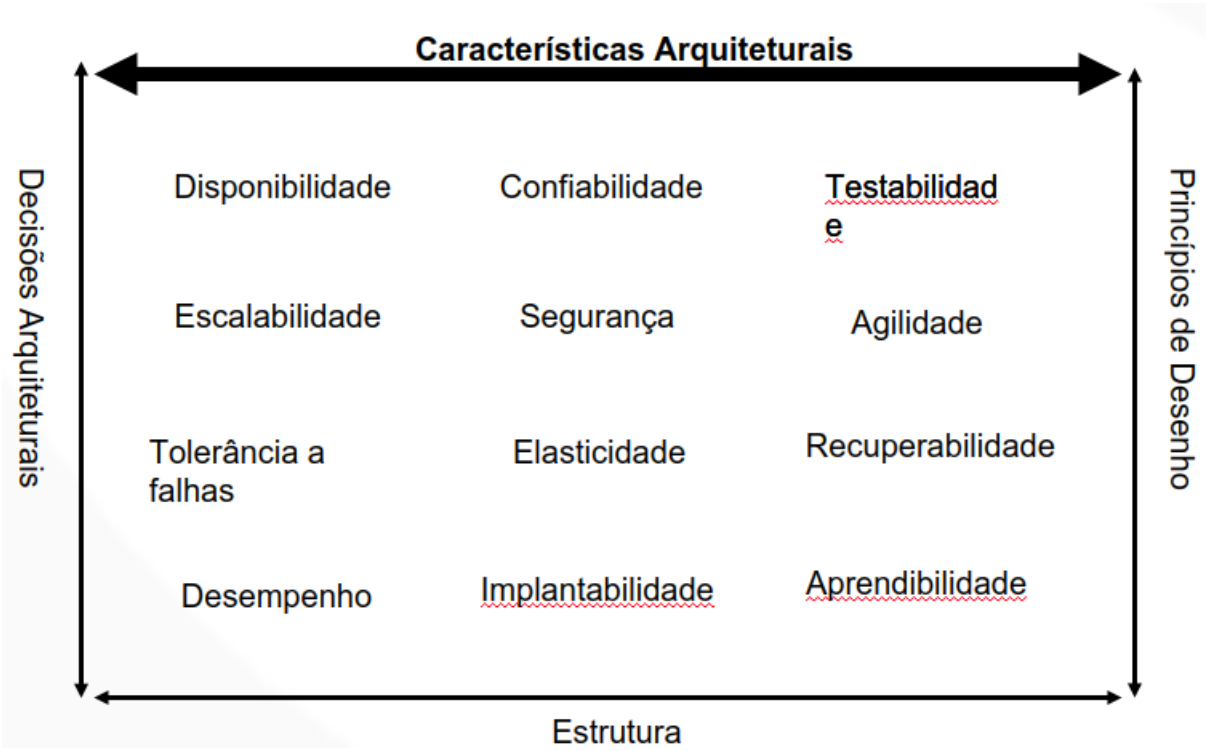
Figura 1 – Elementos da Arquitetura de Software



Fonte: Fundamentals of Software Architecture (RICHARDS, FORD).

Características arquiteturais definem os critérios de sucesso do sistema. O conhecimento de tais características não requer conhecimento das funcionalidades do sistema, ainda que sejam requeridas de forma que o sistema funcione corretamente.

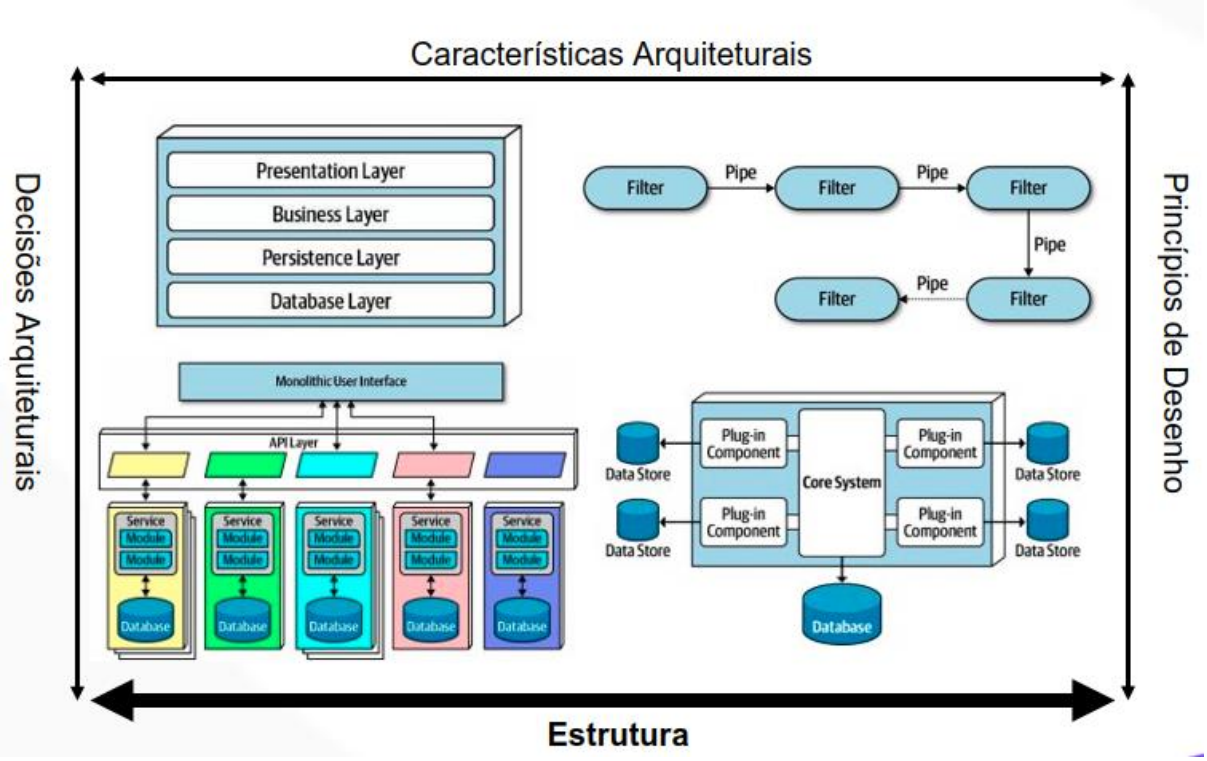
Figura 2 – Características Arquiteturais



Fonte: Fundamentals of Software Architecture (RICHARDS, FORD).

A estrutura se refere ao tipo de estilo arquitetural no qual o sistema é implementado (tais como microsserviços, camadas ou microkernel). A estrutura é um dos primeiros exercícios arquiteturais, visto que provê uma terminologia largamente aceita para um conjunto de características amplo.

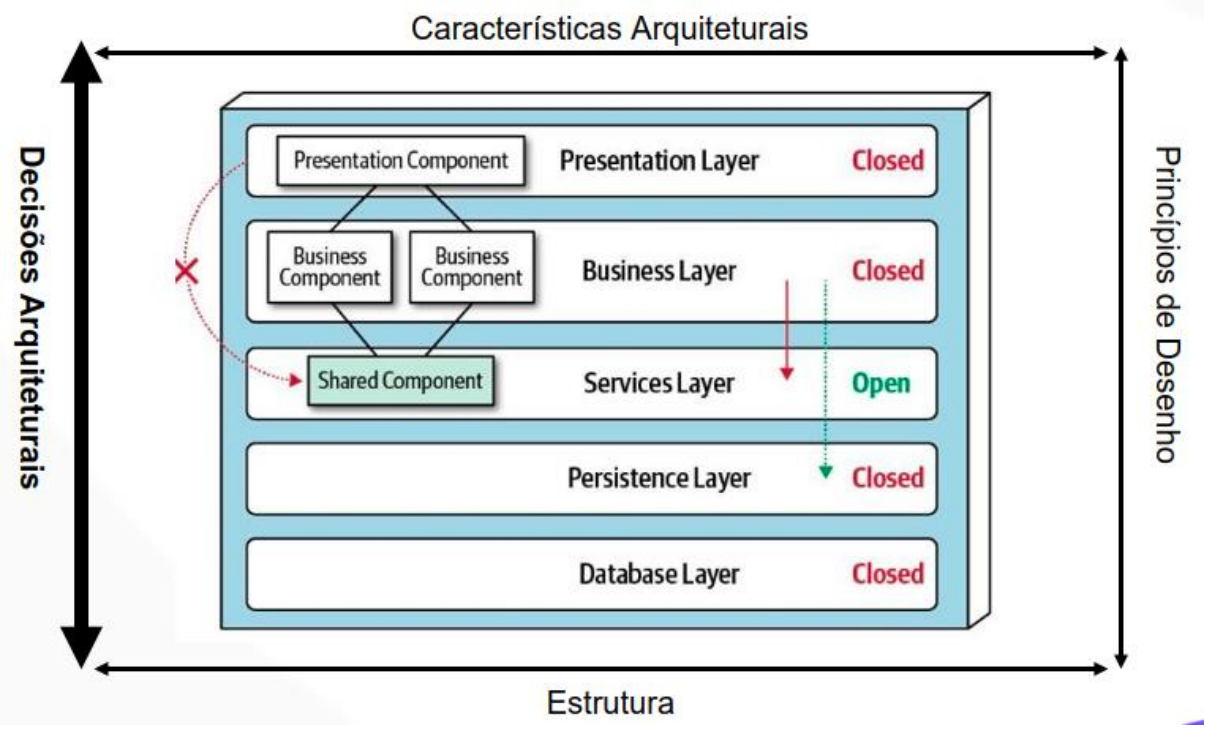
Figura 3 – Estrutura



Fonte: Fundamentals of Software Architecture (RICHARDS, FORD).

A terceira dimensão importante é a das decisões arquiteturais, visto que definem as regras de como um sistema deve ser construído dentro de uma determinada estrutura. Por exemplo, um arquiteto pode tomar uma decisão arquitetural de forma que apenas as camadas de negócio e serviços possam acessar a camada de acesso a dados, restringindo a camada de apresentação de fazer chamadas diretas ao banco de dados. Decisões arquiteturais formam as restrições do sistema e servem de direcionamento para o time de desenvolvimento sobre o que é e o que não é permitido.

Figura 4 – Decisões Arquiteturais

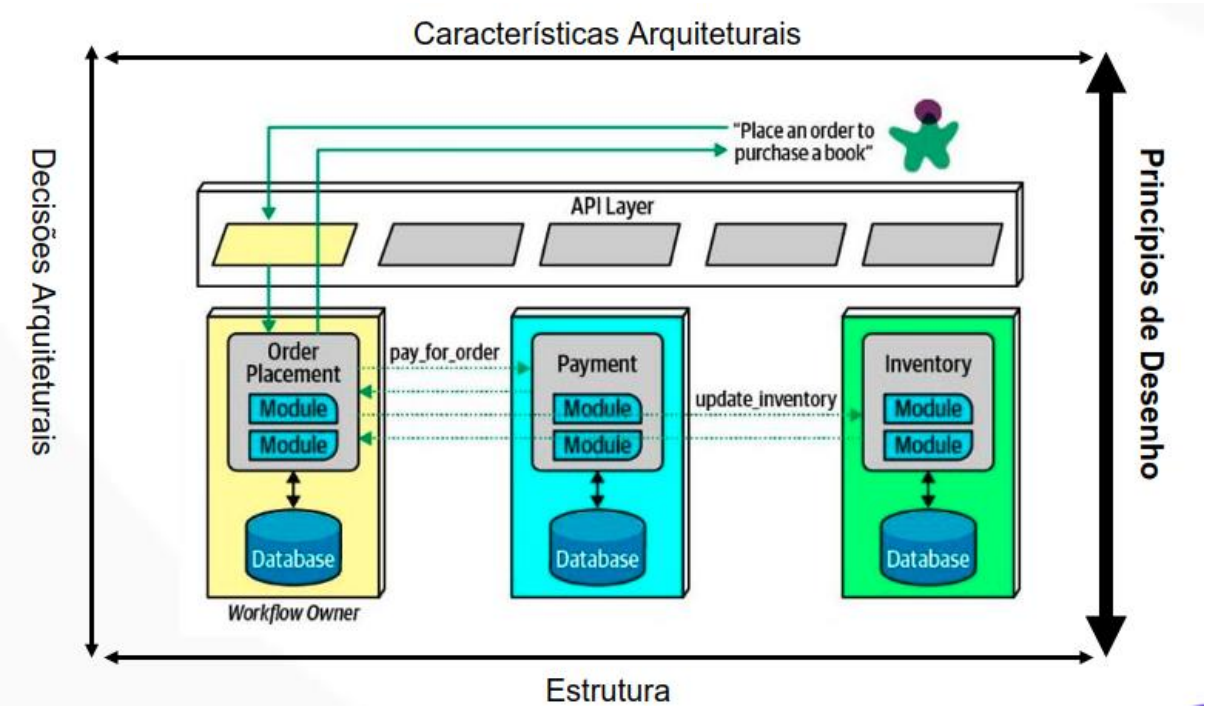


Fonte: Fundamentals of Software Architecture (RICHARDS, FORD).

Por último, mas não menos importante, os princípios de desenho diferem de uma decisão arquitetural na medida que atuam como uma diretriz ao invés de como uma regra mais rígida. Por exemplo, um sistema pode orientar que a troca de mensagens entre serviços dentro de uma arquitetura de microsserviços deve ser feita de forma assíncrona, prioritariamente.

Uma decisão arquitetural nem sempre consegue cobrir todas condições e opções de comunicação entre serviços. Assim, um princípio de desenho pode ser usado para prover orientações sobre o método preferido dada uma circunstância específica.

Figura 5 – Princípios de Desenho



Fonte: Fundamentals of Software Architecture (RICHARDS, FORD).

A coordenação entre essas quatro dimensões permite ter uma visão ao mesmo tempo abrangente e detalhada sobre a arquitetura de uma solução. Nas próximas seções, serão detalhadas as duas dimensões que precisam ser analisadas em primeiro lugar quando se trata de arquitetura de software: as características arquiteturais e os estilos arquiteturais. Mais ao final do capítulo, são mostradas as interseções da arquitetura de software no contexto de desenvolvimento de soluções.

Características Arquiteturais

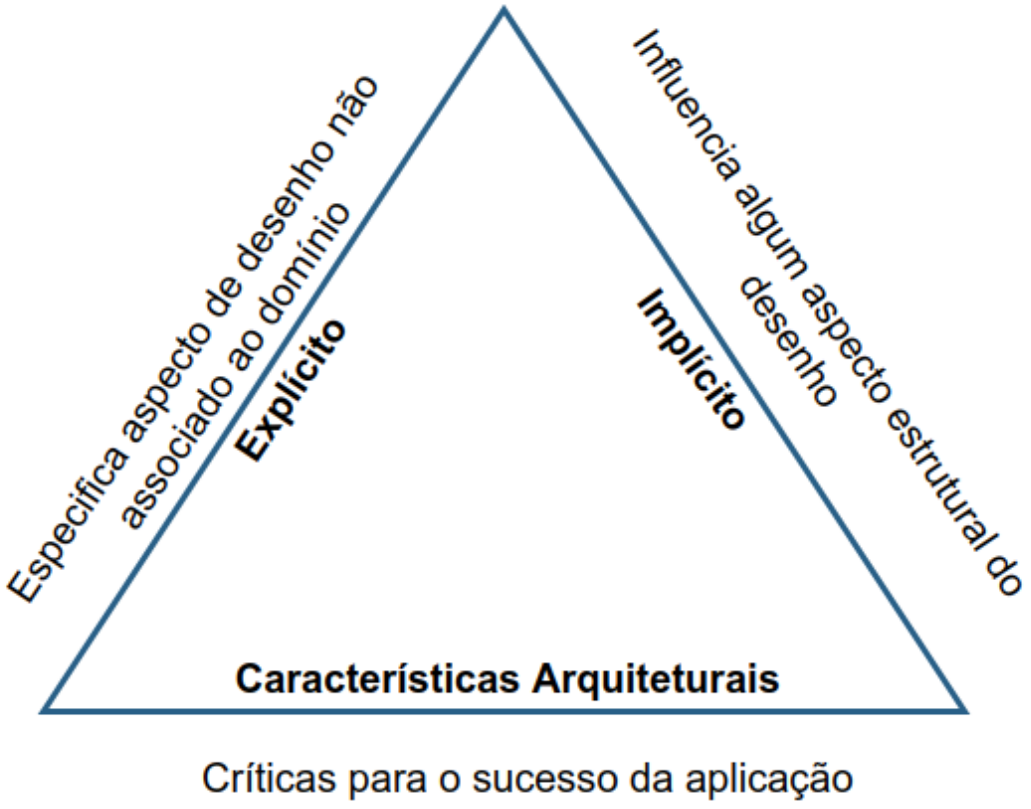
Há diversas fontes de informações que devem ser consideradas nas atividades de arquitetura, incluindo aquelas associadas ao contexto da organização e de negócios como um todo. Entretanto, as informações primordiais estão associadas às características arquiteturais.

Também chamadas de “Requisitos não Funcionais” ou “Requisitos de qualidade”, dizem respeito às características importantes de uma solução, mas que não dependem do domínio do problema.

As características arquiteturais atendem a três critérios (RICHARDS, FORD):

- Especificam considerações de desenho não ligadas ao domínio de negócios;
- Influenciam algum aspecto estrutural do desenho;
- São críticos ou importantes para o sucesso da aplicação.

Figura 6 – Características Arquiteturais



Fonte: Fundamentals of Software Architecture (RICHARDS, FORD).

Seguem alguns exemplos de características arquiteturais, tipicamente encontradas nas aplicações. As definições estão baseadas nas normas da família ISO 25.000.

A **eficiência** de desempenho está entre os requisitos não funcionais mais recorrentes e discutidos nas aplicações em geral. Diz respeito à medida do desempenho relativo à quantidade de recursos usados sob determinadas condições. Requisitos específicos de eficiência de desempenho são:

- Utilização de recursos: "A utilização máxima da CPU deverá ser de 90% em uso";
- Comportamento no tempo: "O tempo de resposta da aplicação deverá ser de 5 segundos para 90% das requisições e 40% de usuários simultâneos."

A confiabilidade é o grau em que um sistema funciona sob determinadas condições por um período de tempo definido. Como requisitos específicos:

- Disponibilidade: "A solução deverá ter disponibilidade 99,9%";
- Tolerância a falhas: "A solução deverá ser mantida em caso de queda de servidores";
- Recuperabilidade: "O tempo médio para recuperação (MTTR) deverá ser de três minutos".

A **segurança** é o grau em que a solução protege informações e dados, de forma que pessoas ou outros produtos ou sistemas tenham o grau de acesso a dados apropriado para os respectivos níveis de autorização. Requisitos específicos:

- Confidencialidade: "Os dados deverão ser acessíveis somente para aqueles autorizados";
- Integridade: "A solução deverá prevenir acesso não autorizado ou modificações no software ou dados";
- Não repúdio: "Ações ou eventos podem ser provados sobre sua origem".

Esta é uma seção apenas introdutória que busca prover uma visão geral dos estilos arquiteturais. O assunto será objeto de análise mais aprofundada em outros módulos.

Após a identificação das características arquiteturais inerentes a uma determinada iniciativa, é necessário buscar enquadrar as necessidades apontadas a partir de um arcabouço pré-definido.

Estilos arquiteturais, ou padrões arquiteturais, descrevem um relacionamento entre componentes que cobrem uma variedade de características arquiteturais.

Um nome de estilo arquitetural captura uma riqueza de detalhes entendidos e descreve a topologia, além das características arquiteturais padrão e assumidas. A existência de uma nomenclatura que remeta a tais informações é primordial para se evitar a "reinvenção" da roda e facilitar a comunicação entre as partes interessadas de um projeto.

Historicamente, há alguns padrões fundamentais que são empregados para entender uma arquitetura (RICHARDS, FORD):

- "*Big ball of mud*": Na verdade não se trata de um padrão, mas da total ausência de estrutura arquitetural minimamente discernível. O termo foi criado por Joseph Yoder (https://joeyoder.com.) e virou uma referência em situações em que simplesmente não é possível identificar o padrão arquitetural de uma solução.

- Arquitetura unitária: Representa historicamente a estrutura primária e original dos primórdios da computação.

- Cliente/Servidor: Primeira iniciativa no sentido de separar a arquitetura unitária original dos computadores. A ideia básica diz respeito à separação entre um ente que provê os serviços e um ou mais consumidores.

A classificação quanto aos padrões fundamentais de arquitetura tem sua relevância na medida que provê uma visão histórica. Entretanto, ao se analisar as características arquiteturais e buscar uma espécie de enquadramento, é necessário se pensar nos tipos de estilos arquiteturais. Essa classificação diz respeito a forma como é feita a implantação (deployment) de uma determinada aplicação em seu ambiente alvo (RICHARDS, FORD):

- Monolítico. Caracteriza-se pelo deployment único de todo código. Os estilos arquiteturais pertencentes a esse tipo são:
  - Arquitetura em camadas;
  - Arquitetura em pipeline;
  - Arquitetura de microkernel.

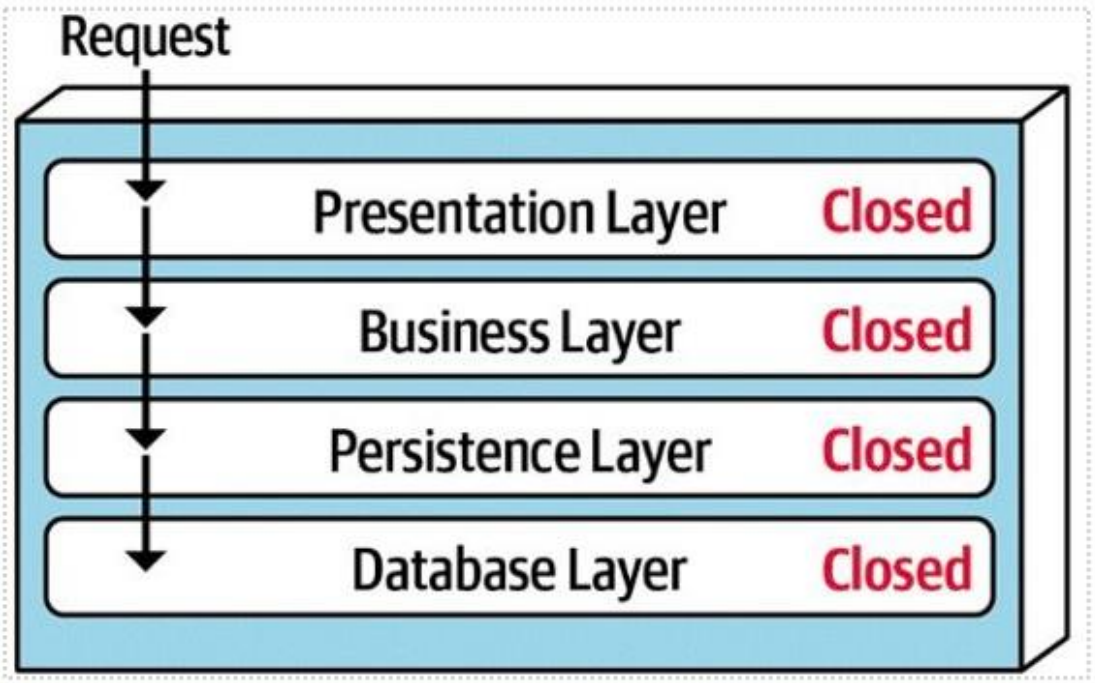
- Distribuído. Caracteriza-se por múltiplas unidades de deployment conectadas por meio de protocolos de acesso remoto. Os estilos arquiteturais pertencentes a esse tipo são:
  - Arquitetura baseada em serviços;
  - Arquitetura baseada em microserviços;
  - Arquitetura baseada orientada a eventos.

A seguir, são descritos alguns desses estilos arquiteturais:

A arquitetura em camadas é o estilo em cuja topologia os componentes são organizados em camadas lógicas horizontais, no qual cada camada desempenha um papel específico na aplicação.

Talvez seja, historicamente, o estilo padrão “de fato” dentre as arquiteturas, dada a sua simplicidade, familiaridade e baixo custo. Os exemplos mais comuns são aplicações web em geral.

Figura 7 – Arquitetura em Camadas

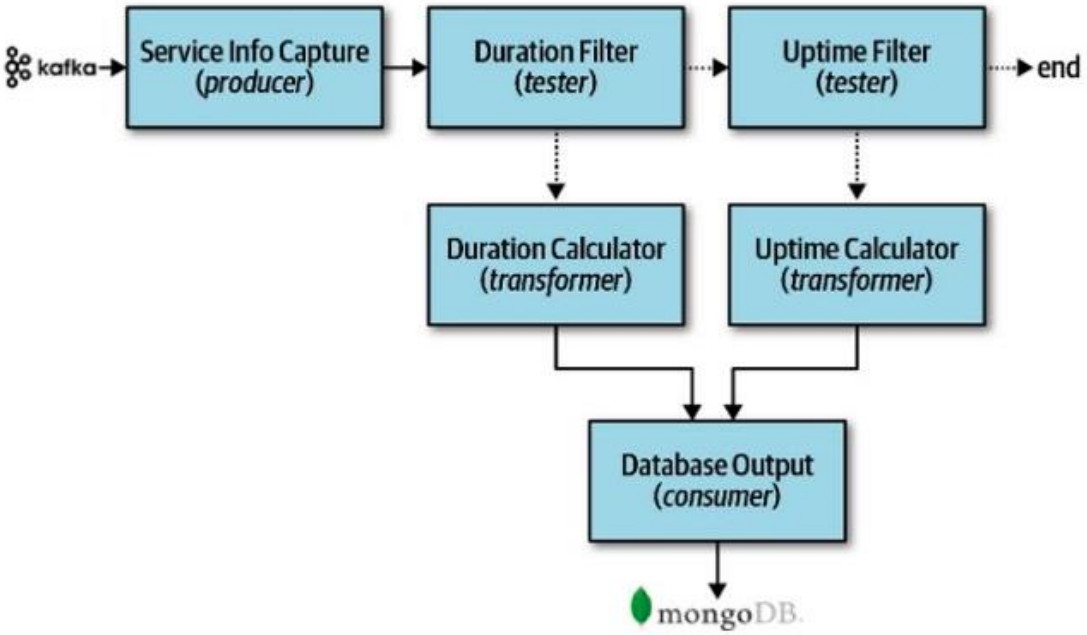


Fonte: Fundamentals of Software Architecture (RICHARDS, FORD).

A arquitetura de pipeline é um estilo cuja topologia consiste nos elementos pipes e filtros. Pipes formam os canais de comunicação entre os filtros. Cada pipe é tipicamente unidirecional e de ponta a ponta. Já os filtros são elementos autocontidos, independentes de outros filtros e são, geralmente, sem estado. Desempenham uma única tarefa, podendo atuar como produtores, transformadores, testadores ou consumidores.

Alguns exemplos de soluções são o pipe dos sistemas Unix e soluções que envolvem extração, transformação e carga de dados (ETL).

Figura 8 – Arquitetura em pipeline

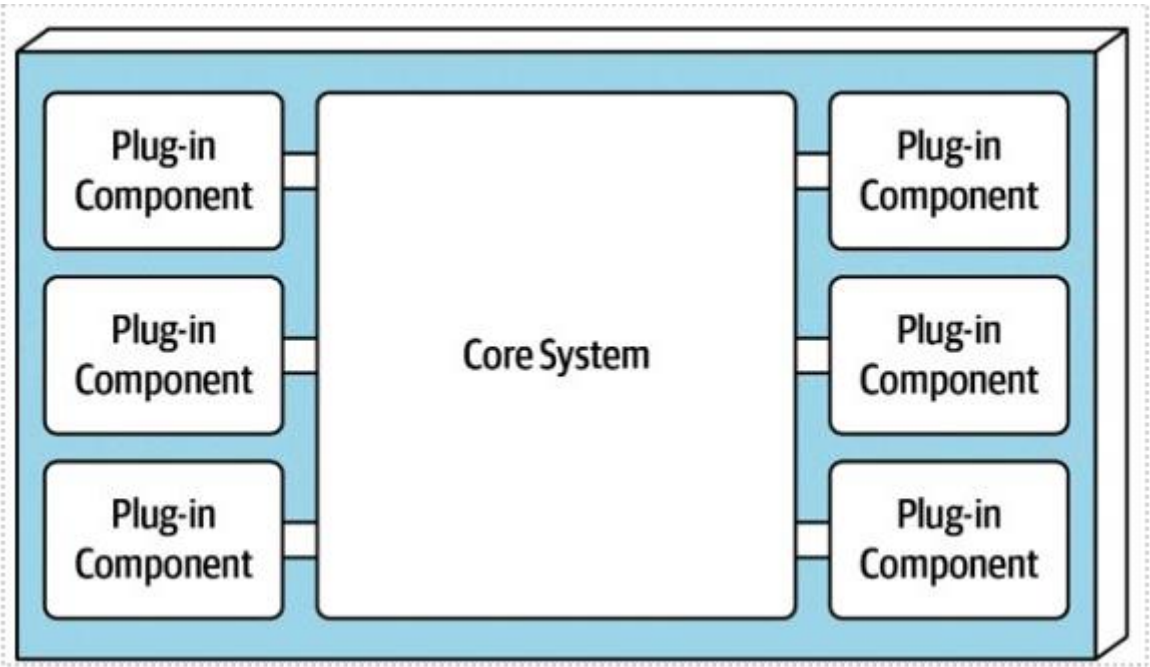


Fonte: Fundamentals of Software Architecture (RICHARDS, FORD).

A arquitetura de microkernel é um estilo cuja topologia consiste em dois elementos: um sistema core, núcleo, e componentes de plug-in. A lógica da aplicação é dividida entre plug-ins independentes e o sistema core básico, provendo extensibilidade, adaptabilidade e isolamento das características da aplicação e lógica de processamento personalizado.

Alguns exemplos de soluções são o *Integrated Development Environment* (IDE) Eclipse e o navegador Google Chrome.

Figura 9 – Arquitetura em microkernel



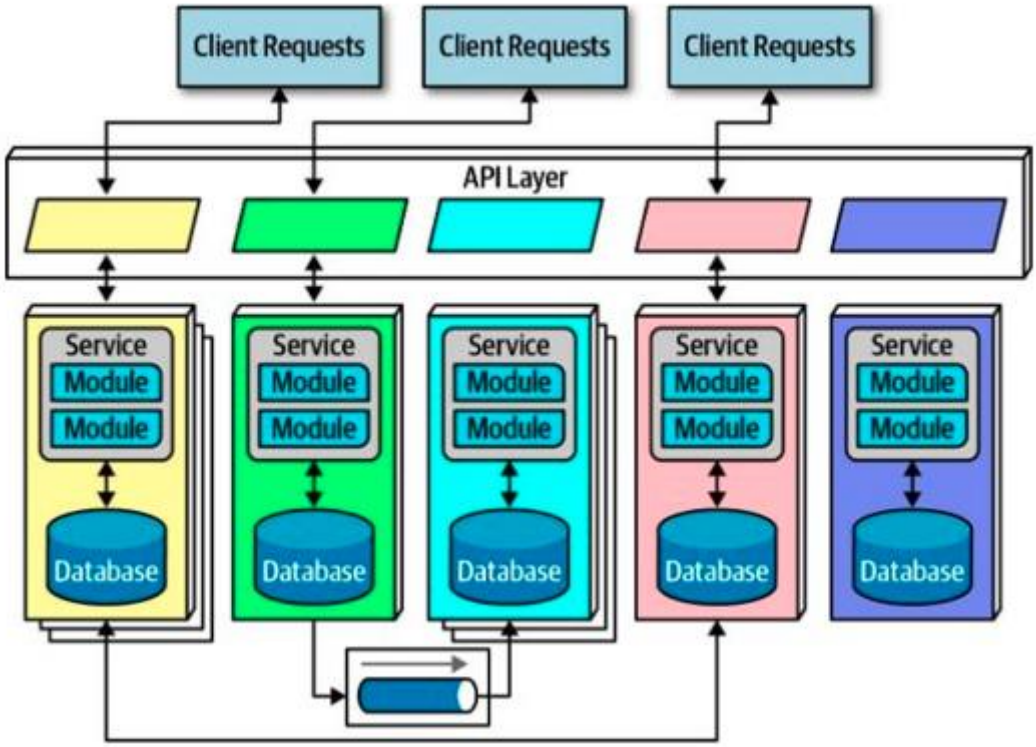
Fonte: Fundamentals of Software Architecture (RICHARDS, FORD).

A arquitetura de microsserviços talvez seja o estilo mais em voga nos últimos anos, muito em função do aumento da complexidade das aplicações, mas também potencializado por abordagens de automação de implantação, caras ao DevOps.

Nesse estilo arquitetural, é esperado que cada serviço inclua todas partes necessárias à sua operação de forma independente, incluindo bancos de dados e outros serviços.

Soluções com esse estilo arquitetural caracterizam-se por permitir o particionamento de contexto de maneira plena, ou seja, um serviço pode ter um alto nível de coesão e baixo nível de acoplamento. Aplicações orientadas a workflows são contextos típicos de aplicação.

Figura 10 – Arquitetura em microsserviços



Fonte: Fundamentals of Software Architecture (RICHARDS, FORD).

O entendimento das vantagens e desvantagens de cada estilo arquitetural é primordial para uma escolha mais adequada a cada contexto. Cada decisão arquitetural é um trade-off, o que acarreta entender que não há uma "decisão arquitetural perfeita". Há apenas uma decisão menos pior. Cabe ao arquiteto uma análise pormenorizada das características arquiteturais e a escolha da solução que tenha maior aderência, entendendo claramente quais são as perdas e ganhos.

A figura a seguir mostra um mapeamento entre algumas características arquiteturais significativas e o grau em que alguns estilos arquiteturais as endereçam.

Figura 11 – Comparativo dos Estilos Arquiteturais

Característica	Camadas	Pipeline	Microkernel	Microsserviços
Tipo de particionamento	Técnico	Técnico	Domínio/ técnico	Domínio
Quantum arquitetural (*)	1	1	1	1 a muitos
Implantabilidade	*	**	***	****
Elasticidade	*	*	*	*****
Modularidade	*	***	***	*****
Desempenho	**	**	***	**
Confiabilidade	***	***	***	****
Simplicidade	*****	*****	****	*
Testabilidade	**	***	***	****

Fonte: Fundamentals of Software Architecture (RICHARDS, FORD).

Cabe destacar aqui o conceito de “quantum arquitetural” criado por Ford, Parsons e Kua e associado ao contexto de arquiteturas evolucionárias. Um quantum arquitetural diz respeito a um artefato implantável (deployable) de forma independente, com alta coesão funcional e conascência (connascence) síncrona (RICHARDS, FORD):

- Implantável de forma independente: inclui todos componentes necessários para funcionar de forma independente de outras partes da arquitetura. Por exemplo, se uma aplicação usa um banco de dados, esse faz parte do quantum porque a aplicação não funciona sem ele.

- Alta coesão funcional: coesão no desenho de componentes refere-se a quão bem o código contido é unificado em propósito. Por exemplo, um componente Cliente com propriedades e métodos todos pertencentes a um Cliente tem alta coesão. Um componente Utilitário com uma miscelânea de recursos possui baixa coesão

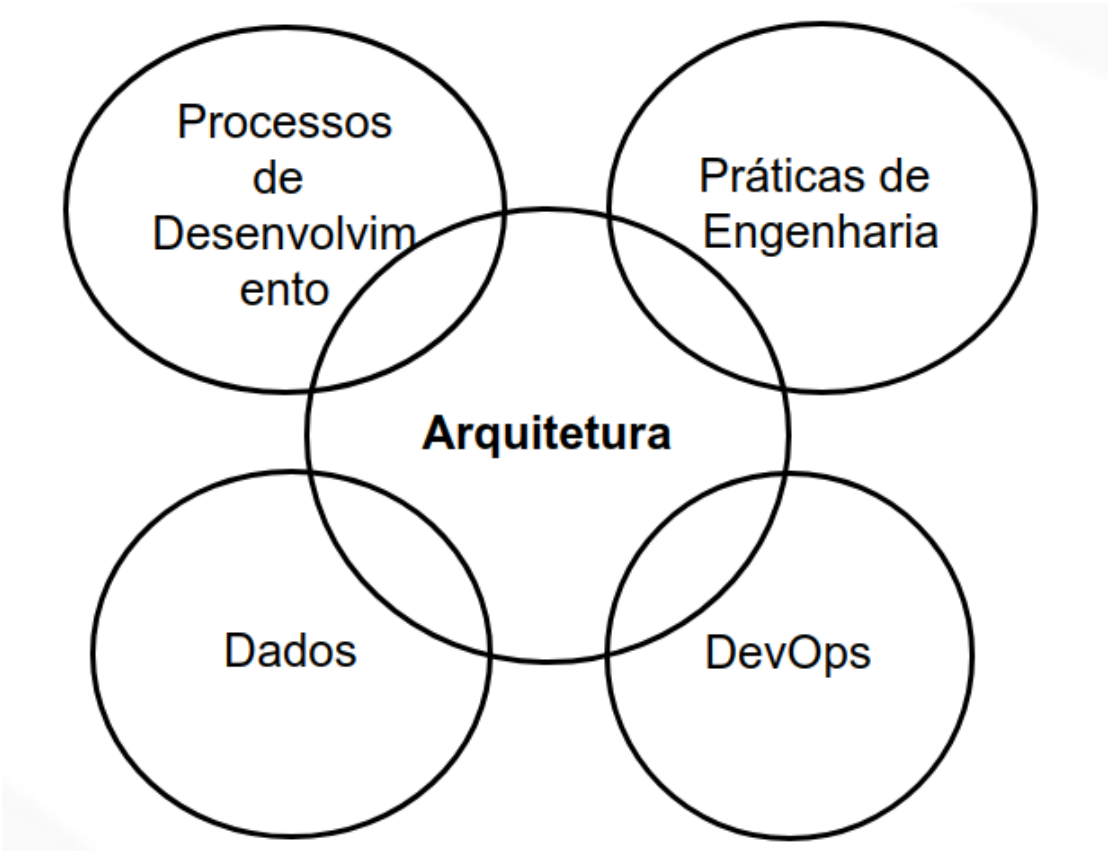
- Conascência síncrona: implica chamadas síncronas dentro do contexto de uma aplicação ou entre serviços distribuídos que formam esse quantum arquitetural. Por exemplo, se um serviço em uma arquitetura de microsserviços chama outro serviço de forma síncrona, cada serviço não pode exibir diferenças extremas nas características arquiteturais operacionais.

Interseções da Arquitetura

Ao longo dos anos, dada a sua importância no contexto organizacional e a gama de respostas que passou a ter que responder, a arquitetura deixou o silo apenas de determinadas soluções tecnológicas e passou a dialogar com outras áreas de conhecimento e com temas relevantes para a construção de soluções.

Em particular, abordagens e práticas relativamente recentes têm se relacionado de maneira cada vez mais efetiva com a arquitetura, o que tem provocado retroalimentações bidirecionais. A forma como as soluções são desenvolvidas, as práticas específicas de engenharia, os bancos de dados e a abordagem Devops, são algumas das interseções de destaque da arquitetura.

Figura 12 – Interseções da Arquitetura



Fonte: Fundamentals of Software Architecture (RICHARDS, FORD).

Os processos de desenvolvimento de soluções têm experimentado mudanças significativas ao longo dos anos. Cada vez mais tem se rendido à realidade complexa, no sentido de não ter uma relação causa e efeito linear, ou seja: um planejamento detalhado no início do projeto não é garantia de sucesso. O surgimento das abordagens iterativas e incrementais e, mais recentemente, da abordagem ágil e do Framework Scrum, em particular, potencializaram mudanças significativas nas abordagens arquiteturais.

O foco em reduzir ao máximo o ciclo de feedback, permitindo obter informações valiosas sobre as hipóteses pensadas para um projeto, tem trazido ganhos significativos também para a arquitetura, que se permite ser mais

experimental e também mais sujeita a repensar e reestruturar arquiteturais, o que no passado era algo pouco provável.

Outra interseção importante da arquitetura é com as práticas de engenharia, notadamente aquelas associadas à abordagem Ágil Extreme Programming (XP). Dada sua ênfase técnica, o XP permitiu focar em práticas como Integração Contínua, Desenvolvimento Orientado a Testes, *Test Driven Development* (TDD) e Refatoração. Essas práticas proveem recursos robustos para a melhoria contínua e antecipada das soluções e verificações constantes dos desenhos e também da arquitetura das soluções.

Uma espécie de reboot das práticas técnicas propostas pela XP foi feita recentemente com o surgimento do Devops. É um movimento profissional que enfatiza a colaboração entre equipes de desenvolvimento e operações visando compatibilizar dois objetivos, em tese contraditórios, que são o foco em entregas mais frequentes e mais rápidas com a estabilidade dos ambientes. A arquitetura tem sido fundamental nesse movimento, visto que pode prover recursos importantes para a liberação contínua, como as *features flags*, as liberações canário e arquiteturas como a de microsserviços, por exemplo. Por outro lado, a profundidade de análise arquitetural foi repensada. Anteriormente, as soluções arquiteturais tinham que prover soluções para questões "caixa preta" típicas da área de operações, como escalabilidade e elasticidade, embutindo considerável complexidade. Com a maior colaboração, o pessoal de operações passou a se envolver mais e assumir a busca de soluções para essas questões.

Por último, uma interseção fundamental diz respeito aos bancos de dados, recurso inseparável da maioria das soluções corporativas. À exemplo do que ocorre no contexto Devops, aqui também o nome do jogo é "colaboração". Nesse caso, a colaboração mais efetiva entre o arquiteto e os profissionais especializados em bancos de dados, tais como administradores de dados (ADs) e Administradores de

Banco de dados (DBAs), no sentido de escolher arranjos mais adequados a um determinado contexto.

Tal ação alçou os bancos de dados ao patamar de critério primordial na escolha e adaptação de uma determinada arquitetura. A arquitetura de microsserviços, por exemplo, traz como premissa o fato de que os serviços devem conter todos os recursos necessários para sua execução. Os bancos de dados são inevitáveis nesse contexto.

Considerações finais

Este capítulo buscou prover uma visão geral sobre o tema “arquitetura”, trazendo uma reflexão sobre seu conceito e seus principais componentes, como as características, estilos, decisões e padrões de desenho. Foram apresentadas, ainda, algumas das interseções que mais influenciaram as evoluções arquiteturais ao longo dos anos.

Capítulo 2. O Papel do Arquiteto de Software

Na mesma medida em que as soluções tecnológicas ganharam força no contexto empresarial, também aumentou a importância e a demanda pela atuação do arquiteto. Se antes havia uma clara alocação do arquiteto como membro de um clube restrito ou um silo cheio de termos técnicos, hoje é necessário dialogar com várias tribos fora de uma única equipe, por toda a organização e até mesmo fora dela. Ao mesmo tempo, o trabalho colaborativo inerente às práticas ágeis alterou consideravelmente a atuação do arquiteto dentro das equipes.

Uma visão importante sobre a atuação do arquiteto é trazida por Martin Fowler: “O arquiteto deve ser como um guia, um experiente e capacitado membro da equipe que ensina aos outros se virar melhor, e ainda assim está sempre lá para as partes mais complicadas”.

Expectativas com relação ao papel do arquiteto

A exemplo do que acontece com toda demanda por conhecimento da área, em particular para os arquitetos, ao longo dos anos o perfil mudou de algo extremamente técnico para algo que também demanda um conhecimento de negócio.

Tem-se utilizado muitos rótulos sobre o “perfil de um arquiteto”. São recorrentes discussões como a histórica polarização entre soft e hard skills e a avaliação de competências por meio das dimensões conhecimentos, habilidades e atitudes.

De uma forma mais geral, podem ser elencadas algumas expectativas com relação ao papel do arquiteto da seguinte forma (RICHARDS, FORD):

- Tomar decisões arquiteturais;
- Analisar a arquitetura continuamente;
- Manter-se atualizado com as tendências mais recentes;
- Garantir conformidade com decisões;

- Diversificar entre exposição e experimentação;
- Ter conhecimento no domínio do negócio;
- Possuir habilidades interpessoais;
- Entender e navegar na política.

Tomar decisões arquiteturais

A tomada de decisões arquiteturais talvez esteja entre as principais expectativas com relação ao arquiteto. É fundamental entender que, mais do que ser o único tomador de decisão (de forma monocrática), o arquiteto deve ser aquele que foca em guiar a tomada de decisão. Por exemplo, ao invés de definir um determinado framework ou tecnologia para o desenvolvimento mobile, o arquiteto pode compilar os critérios para uma tomada de decisões efetiva.

Outro aspecto importante da tomada de decisões é entender o papel cada vez mais importante do arquiteto, visto que as decisões podem ser pensadas em vários níveis, desde o time até o departamento ou toda a organização.

Analisar a arquitetura continuamente

Prover mecanismos e avaliar a arquitetura continuamente é uma expectativa fundamental para manter a relevância das soluções, além de um grande desafio, visto que nem sempre há essa previsão na alocação da disponibilidade do arquiteto por períodos mais longos em uma iniciativa.

Uma arquitetura com vitalidade é algo que ainda não é percebido como algo de valor, exceto nos casos em que a solução atual apresenta problemas muito graves, quando já pode ser tarde demais.

Manter-se atualizado com as tendências mais recentes

A área de tecnologia é profícua não só em mudanças, mas naquelas feitas em ritmos bastante acelerados. O desafio para o arquiteto é consideravelmente maior

do que para outros profissionais, visto que a capacidade de se manter atualizado é o que garante arquiteturas que durem por mais tempo.

Não se manter alinhado às tecnologias mais recentes e às tendências da indústria pode tornar obsoletos tanto as soluções arquiteturais quanto os arquitetos que as tomam.

Garantir conformidade com decisões

Na mesma linha de dificuldade de garantir continuamente a vitalidade e a atualidade, garantir que as decisões tomadas em determinado momento sejam cumpridas é um fator desafiador. Verificar continuamente se o time de desenvolvimento está seguindo as decisões arquiteturais e princípios de desenho definidos, documentados e comunicados pelo arquiteto deve fazer parte da rotina do arquiteto.

Diversificar entre exposição e experimentação

Há um dilema recorrente para todo profissional que é avaliar se vale mais a pena ser generalista ou especialista. Para o arquiteto, essa discussão não ajuda muito, visto que ele acaba sendo obrigado a equilibrar as duas coisas. Diversificar tanto a exposição quanto a experimentação é fundamental, e deve ser algo buscado focando em amplitude técnica ao invés de profundidade técnica. Amplitude inclui as coisas que o arquiteto conhece, mas não em um nível detalhado, combinado com as coisas que ele conhece detalhadamente.

Ter conhecimento no domínio do negócio

Já foi o tempo em que o arquiteto poderia ser vangloriar de não conhecer o domínio de negócios, o que para alguns era até algo de certo demérito. Certamente, o arquiteto não precisa e, em geral nem pode, ser o maior especialista em negócios de um determinado domínio. Em contrapartida, deve-se sempre buscar conhecimento dos aspectos mínimos do domínio.

Isso envolve melhorar a capacidade entender os problemas, objetivos e requisitos, de forma a desenhar uma arquitetura efetiva para atender aos requisitos de negócio. Além disso, deve-se melhorar a capacidade de comunicação do arquiteto com as partes interessadas no projeto, aumentando sua credibilidade.

Possuir habilidades interpessoais

Fala-se muito em habilidades interpessoais para um arquiteto. De fato, considerando que nenhuma solução é feita de forma individualizada, isso é inevitável. O arquiteto não precisa se tornar um profissional de ciências humanas. Por outro lado, ele só consegue viabilizar seu trabalho exercitando melhor algumas características como comunicação, o trabalho em grupo, a facilitação, a liderança e a negociação.

Entender e navegar na política

Uma espécie de capacidade interpessoal complementar está associada com a capacidade de se entender e navegar na política da organização. Isso porque, entender o contexto político e a forma como as decisões potenciais podem afetar a determinadas partes interessadas. Saber utilizar técnicas de negociação para ter as decisões aprovadas também é algo associado a esse contexto.

Na próxima seção, serão apresentados alguns aspectos complementares à atuação do arquiteto, especialmente a forma como encarar determinadas questões.

Mentalidade de Arquitetura

Arquitetura é um assunto que traz a reboque algumas discussões mais elaboradas, conforme mostrado anteriormente. De uma maneira geral, além das expectativas já citadas, há algumas questões que moldam a mentalidade de um arquiteto (RICHARDS, FORD):

- Estabelecer interfaces entre arquitetura e desenho;
- Compatibilizar visão ampla versus visão profunda;

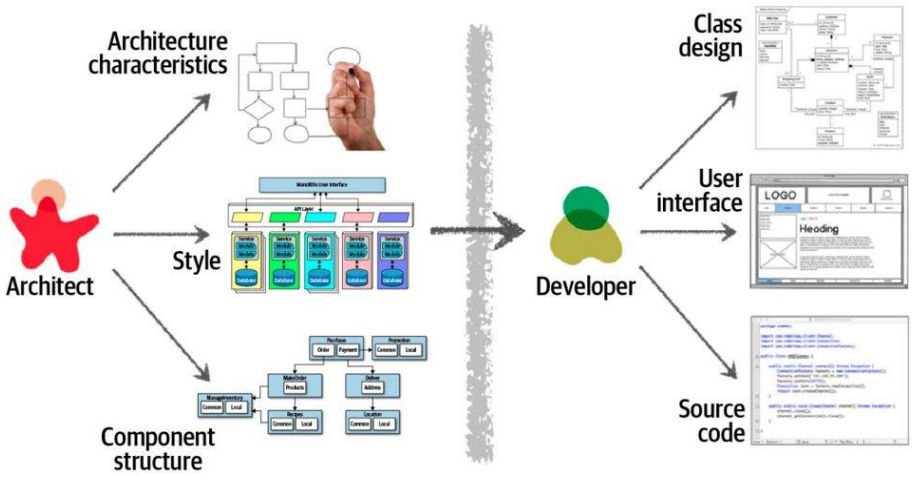
- Analisar trade-offs arquiteturais;
- Entender os fatores direcionadores de negócio.

Estabelecer interfaces entre arquitetura e desenho

Historicamente, tem-se discutido as diferenças entre arquitetura e desenho. Para alguns autores há uma separação clara, para outros nem tanto. De uma mentalidade mais localizada, fica clara a separação como sendo dois silos distintos, com um contrato claro entre as “áreas".

Essa separação acaba por acarretar um fato consumado: não funciona. Essa segregação do arquiteto em uma "Torre de Marfim” acaba apenas por afastá-lo da realidade dos projetos, o que acaba criando um vale.

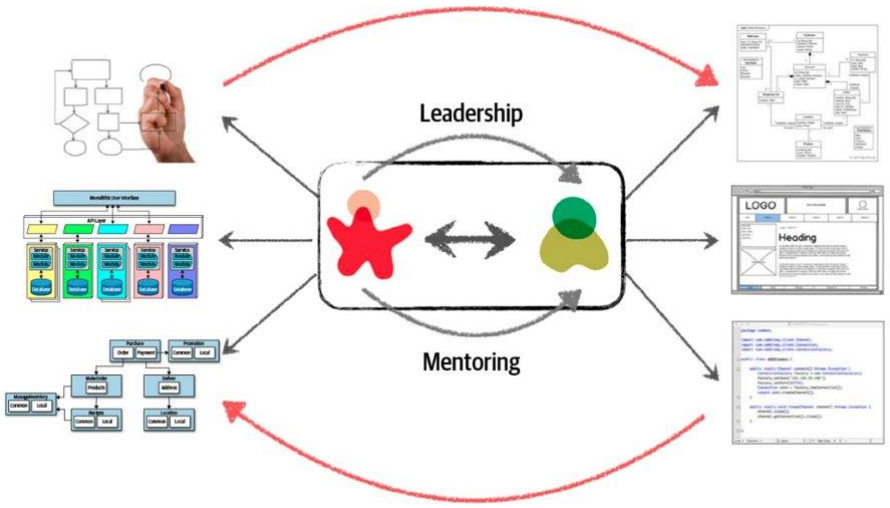
Figura 13 – Interface tradicional entre arquitetura e desenho



Fonte: Fundamentals of Software Architecture (RICHARDS, FORD)

O fato é que, mais relevante do que tratar semelhanças e diferenças, é entender a natureza iterativa da fronteira entre as duas frentes. Uma forma mais efetiva é pensar em um processo mais colaborativo, com o arquiteto assumindo mais um papel de liderança e mentoria.

Figura 14 – Interface repensada entre arquitetura e desenho

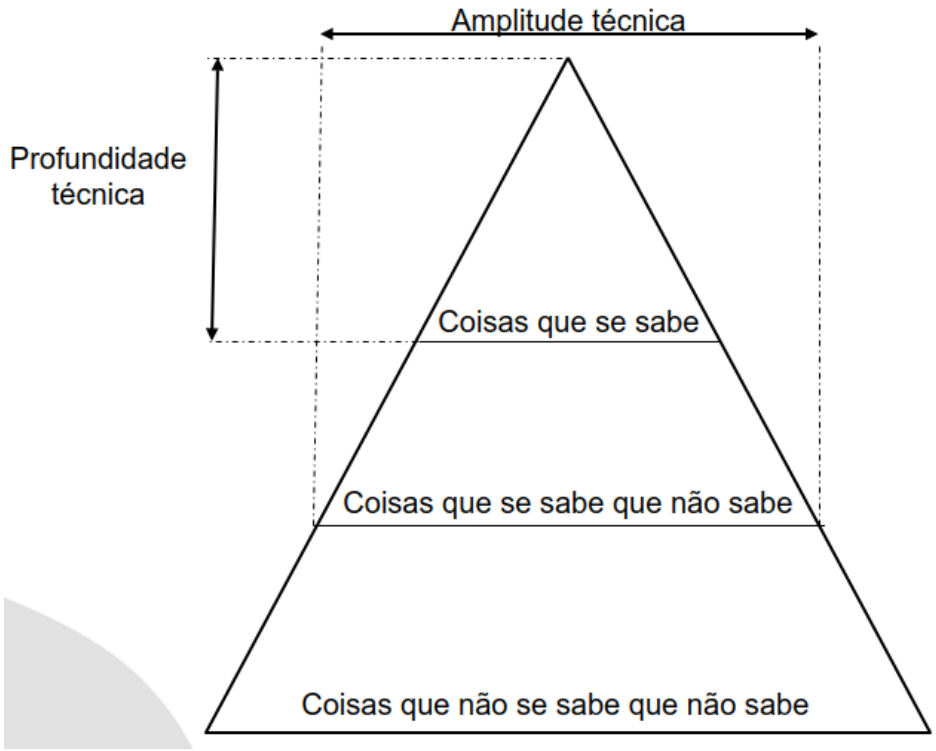


Fonte: Fundamentals of Software Architecture (RICHARDS, FORD)

Compatibilizar visão ampla versus visão profunda

Uma das expectativas com relação ao arquiteto é equilibrar profundidade é abrangência. Isso pode ser obtido entendendo-se e separando-se claramente as fronteiras daquilo que se sabe e daquilo que se sabe que não sabe.

Figura 15 – Níveis de Conhecimento



Fonte: Fundamentals of Software Architecture (RICHARDS, FORD)

Entender os fatores direcionadores de negócio

O arquiteto tem sido alçado à posição de "arquiteto corporativo" em vários contextos organizacionais, o que o faz mudar de interlocução e pensar outras dimensões além da técnica.

As características arquiteturais, sendo um ponto de partida eminentemente técnico, devem ser compatibilizadas com os respectivos direcionadores de negócio. É possível fazer uma espécie de "de para" no sentido de entender as prioridades de acordo com cada contexto organizacional.

Figura 16 – Direcionadores de Negócio e Arquitetura

Direcionador de Negócio	Características Arquiteturais
Fusões e aquisições	Interoperabilidade, escalabilidade, adaptabilidade, extensibilidade
"Time to market"	Agilidade, testabilidade, implantabilidade
Satisfação do usuário	Desempenho, disponibilidade, tolerância a falhas, testabilidade, implantabilidade, agilidade, segurança
Vantagem competitiva	Agilidade, testabilidade, implantabilidade, escalabilidade, disponibilidade, tolerância a falhas
Tempo e orçamento	Simplicidade, viabilidade

Fonte: Fundamentals of Software Architecture (RICHARDS, FORD)

Considerações finais

Este capítulo buscou prover elementos para o entendimento do que seria um perfil típico de um arquiteto e daquilo que embasa uma abordagem ou mentalidade arquitetural. Apesar dos desafios inerentes, cabe destacar também que essas considerações são contextuais, ou seja, dependem do contexto de cada projeto e de cada organização.

Capítulo 3. Abordagens de Desenvolvimento

As abordagens de desenvolvimento dizem respeito às várias formas de se encarar a missão de construir e manter soluções desde sua criação até a descontinuidade. Ao longo dos anos, várias mudanças nas formas de se construir software têm afetado também a forma como a arquitetura é pensada.

Se no passado arquitetura era quase uma visão estática, congelada no tempo, hoje a arquitetura pode ser enxergada por um prisma evolucionário.

Modelos de Ciclos de Vida

Ciclo de vida é algo que existe em vários contextos. O ser humano, por exemplo, tem um ciclo de vida, que vai desde sua concepção, passando pelo seu nascimento até sua morte. Em contexto de produtos ou soluções, um ciclo de vida define as fases que conectam o início e o fim de um produto ou projeto.

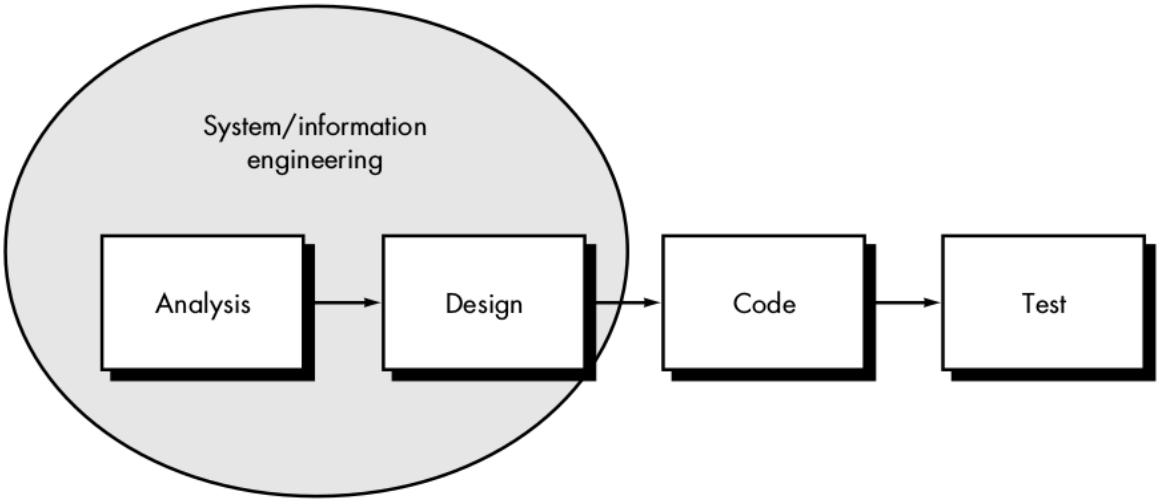
Modelos de ciclo de vida, por sua vez, são referências que servem como uma definição de alto nível das fases que ocorrem durante o ciclo de vida. Não tem como objetivo prover informações detalhadas, somente destacar a atividades-chave e suas interdependências. Alguns exemplos de modelos de ciclo de vida são:

- Cascata;
- Espiral;
- Incremental.

Modelo Cascata

O modelo de ciclo de vida Cascata, também chamado ciclo de vida Clássico ou modelo Linear, sugere uma abordagem sistemática e sequencial para o desenvolvimento de software, começando no nível de sistema e progredindo pelas fases de análise, desenho, codificação, testes e suporte.

Figura 17 – Modelo Cascata



Fonte: Engenharia de Software (PRESMAN).

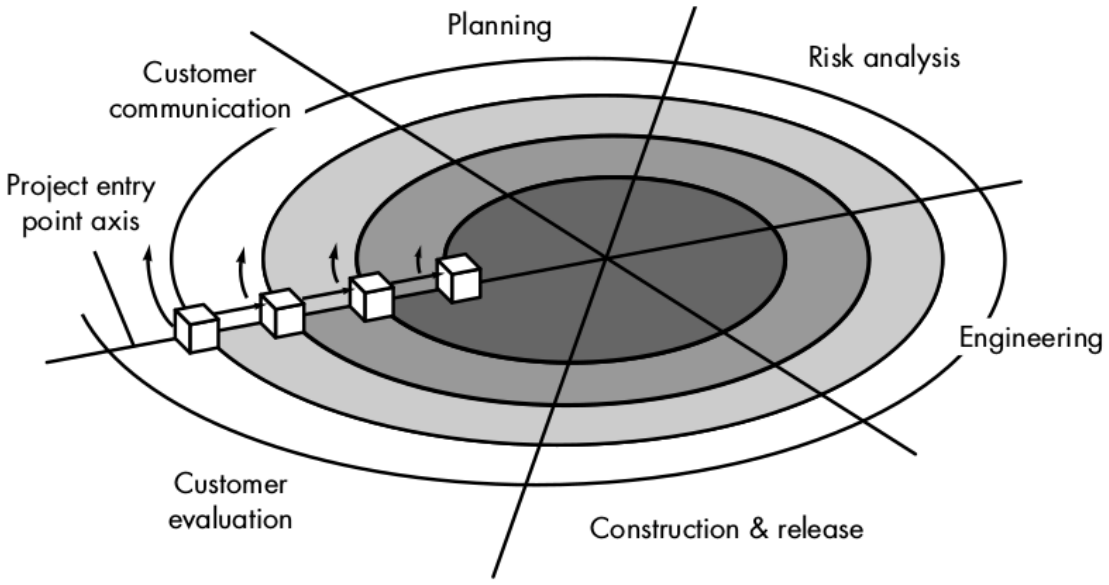
O modelo Cascata é o mais antigo e, até recentemente, o mais usado paradigma de desenvolvimento para engenharia de software. Alguns problemas usualmente atribuídos ao modelo são (PRESSMAN):

- Projetos reais raramente seguem o fluxo sequencial;
- É usualmente difícil para o cliente estabelecer todos requisitos explicitamente, especialmente nas fases iniciais;
- O cliente deve ter paciência, visto que uma versão funcional do programa só estará disponível ao final do projeto.

Modelo Espiral

Modelo evolucionário de software que acopla a natureza iterativa de prototipação com os aspectos controlados e sistemáticos do modelo sequencial ou linear. Provê potencial para um desenvolvimento rápido de versões incrementais de software. O modelo é dividido em atividades e regiões.

Figura 18 – Modelo Espiral

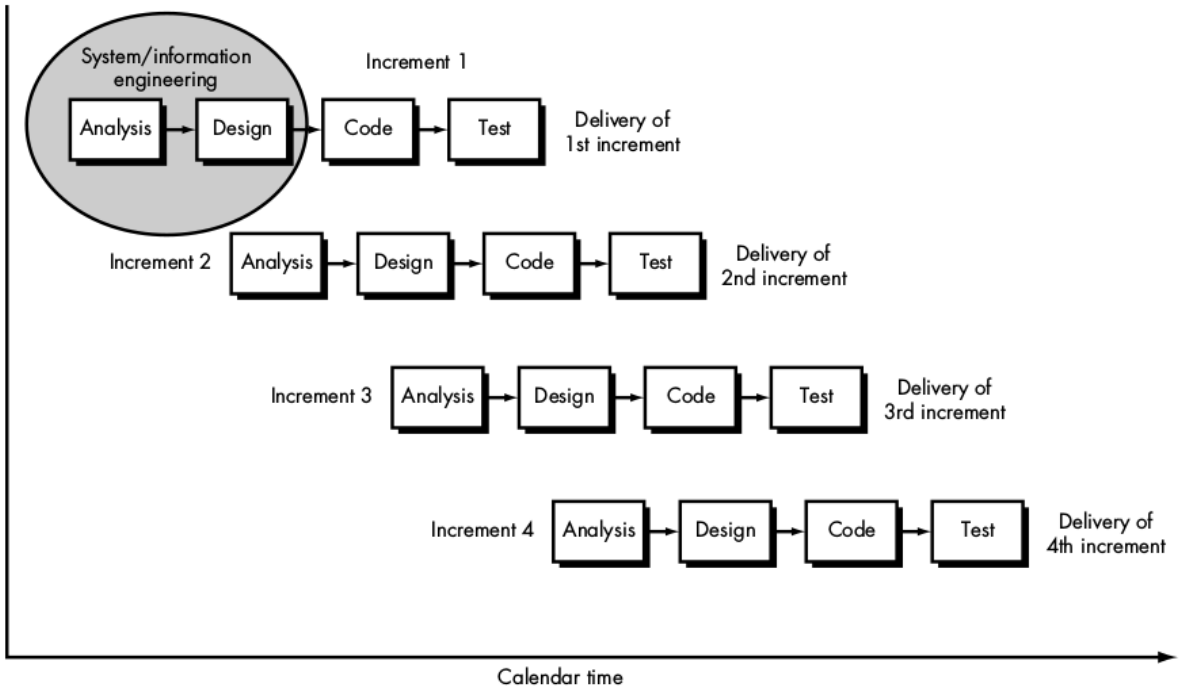


Fonte: Engenharia de Software (PRESSMAN).

Modelo Incremental

Combina elementos do modelo sequencial linear com uma filosofia iterativa. A visão linear é obtida de uma forma estagiada à medida em que o tempo de calendário passa. Cada sequência linear produz uma entrega incremental utilizável de software.

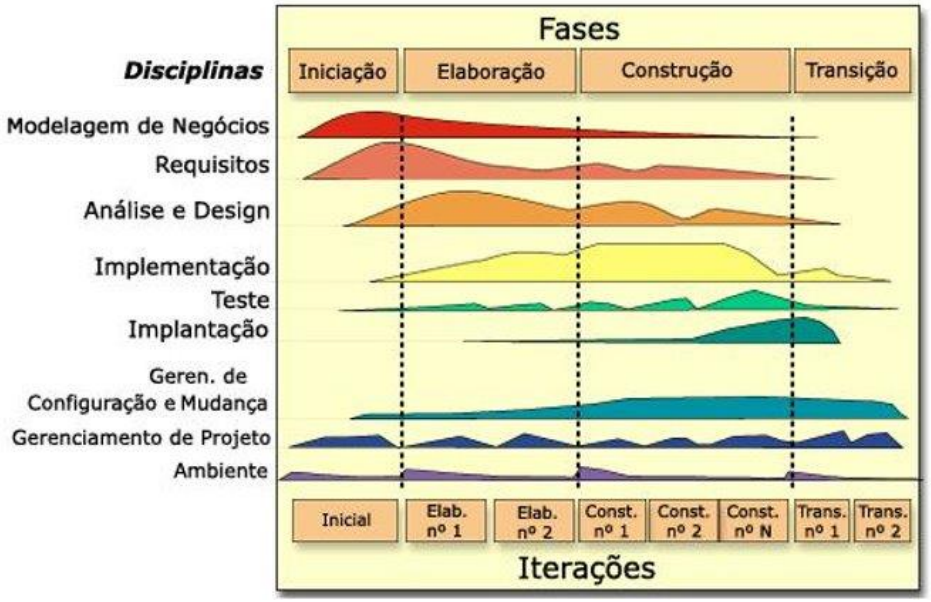
Figura 19 – Modelo Incremental



Fonte: Engenharia de Software (PRESSMAN)

O *Rational Unified Process* (RUP) é ao mesmo tempo um produto e um processo que implementa um ciclo de vida iterativo e incremental. Tornou-se popular no início dos anos 2000, capitaneado pela empresa Rational. É organizado em fases decompostas em iterações. Em cada fase e iteração são realizadas tarefas relativas às disciplinas diversas de engenharia e/ou gestão

Figura 20 – Rational Unified Process (RUP)



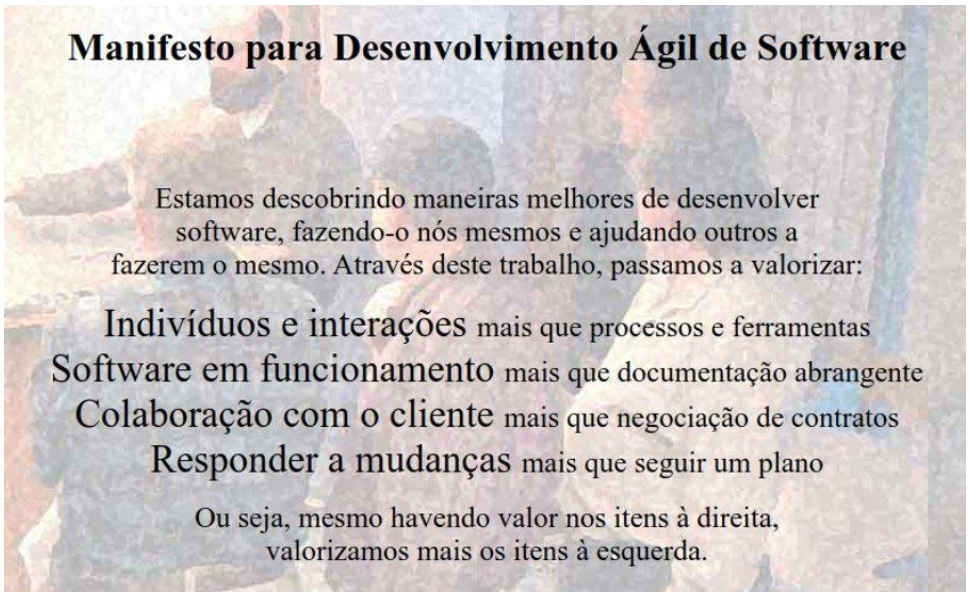
Fonte: Engenharia de Software (PRESSMAN)

Abordagem Ágil

No início dos anos 2000, havia uma ênfase nos processos e ferramentas como forma de se entregar soluções de qualidade. Foi a era de ouro de abordagens como o *Capability Maturity Model Integrated* (CMMI), as normas ISO (especialmente a ISO 900) relativas à qualidade de serviços, e o próprio RUP, citado anteriormente.

Essa ênfase levou um grupo de consultores de software a promover um evento para refletir sobre os rumos da área e buscar formas alternativas de se desenvolver soluções. Com isso, nasceu o Manifesto para Desenvolvimento Ágil de Software, no ano de 2001.

Figura 21 – Manifesto Ágil



Fonte: Agile Manifesto [www.agilemanifesto.org]

O Manifesto Ágil, como ficou conhecido, é detalhado por meio de 12 princípios, conforme disponível em www.agilemanifesto.org:

1. Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor;

2. Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas;

3. Entregar software funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos;

4. Pessoas relacionadas à negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto;

5. Construir projetos ao redor de indivíduos motivados, dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho;

6. O Método mais eficiente e eficaz de transmitir informações para e por dentro de um time de desenvolvimento é através de uma conversa cara a cara;

7. Software funcional é a medida primária de progresso;

8. Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter indefinidamente passos constantes;

9. Contínua atenção à excelência técnica e bom design aumentam a agilidade;

10. Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito;

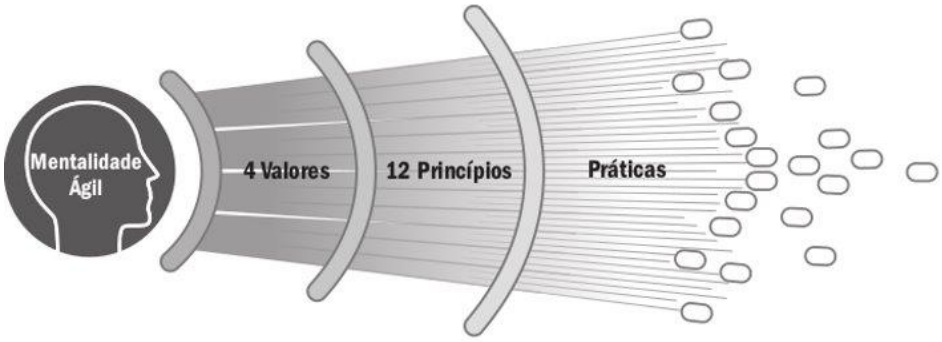
11. As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis;

12. Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

O evento representou um marco, e a utilização do termo “agilidade” passou a ter cada vez mais aceitação na comunidade de desenvolvimento de soluções.

Com o passar o tempo, o próprio termo "agilidade" precisou ser revisitado. Por exemplo, o PMI incorporou essa abordagem em um de seus guias: “Agilidade é uma mentalidade definida por valores, orientada por princípios e manifestada por meio de muitas práticas diferentes” [PMI]. Os praticantes ágeis selecionam as práticas com base em suas necessidades.

Figura 22 – Mentalidade Ágil

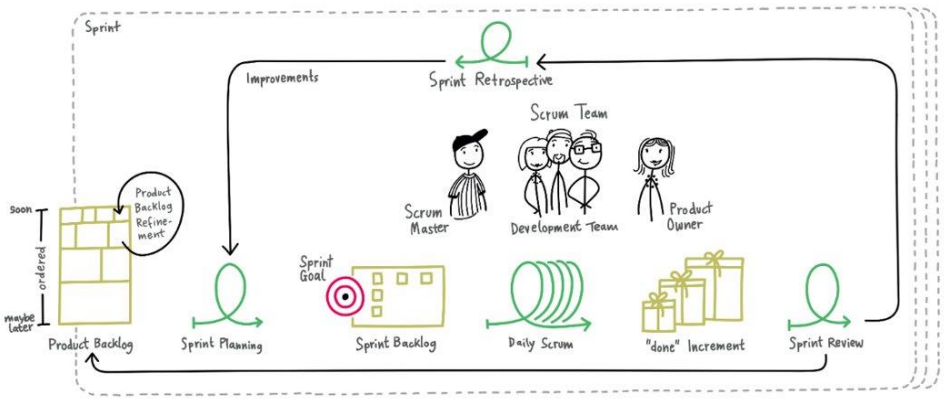


Fonte: Agile Practice Guide [PMI]

Framework Scrum

Considerando o ecossistema de práticas ágeis, a abordagem mais popular é o Scrum, que é um framework para desenvolver e manter produtos complexos. Com o Scrum, as pessoas podem tratar e resolver problemas complexos e adaptativos, enquanto produtiva e criativamente entregam produtos com o mais alto valor possível O Scrum permite empregar vários processos ou técnicas (SCHWABER, SUTHERLAND).

Figura 22 – Framework Scrum



Fonte: https://www.scrum.org/resources/blog/alternative-scrum-framework-poster.

O framework Scrum possui os seguintes componentes principais:

- Pilares: transparência, inspeção e adaptação;
- Papéis: product owner, scrum master e time de desenvolvimento;
- Eventos: planejamento da sprint, reunião diária, reunião de revisão e reunião de retrospectiva;
- Artefatos: backlog do produto, backlog da sprint e incremento.

Capítulo 4. DevOps

Contexto do DevOps

O contexto de negócios atual acarreta uma pressão para as pessoas envolvidas no desenvolvimento de soluções, devido às demandas cada vez maiores por soluções entregues de maneira mais rápida, mais baratas e melhores.

O fluxo de entrega de soluções sofre impactos diretos dessa pressão, visto que, em geral, as organizações não veem de fato um fluxo, mas áreas ou silos com objetivos e incentivos diferentes.

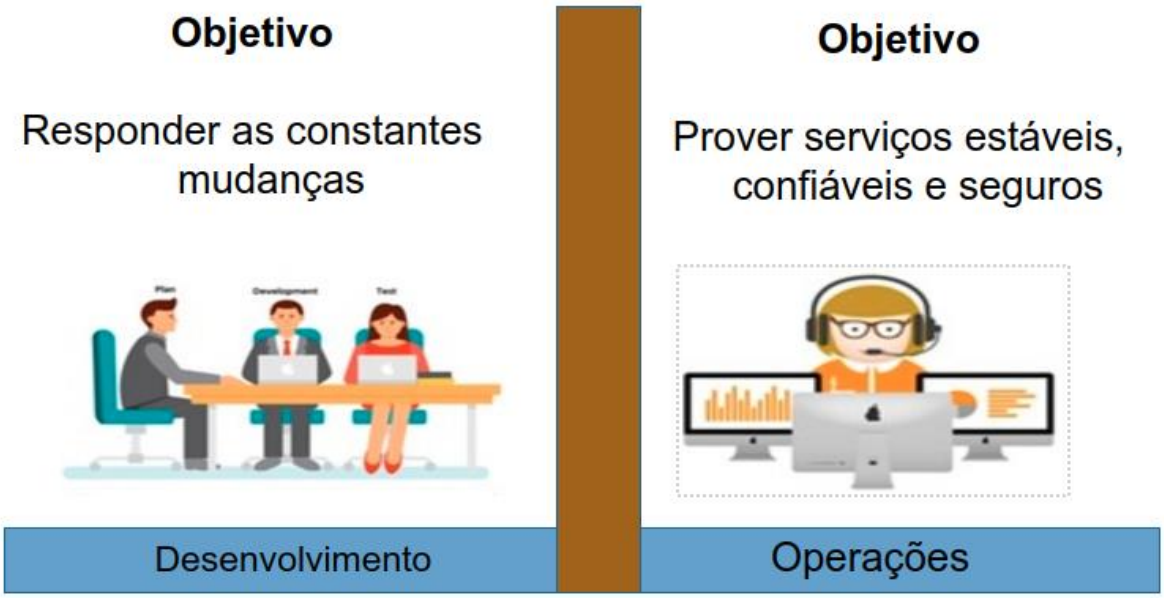
Figura 23 – Fluxo de Entrega de Soluções

Fonte: https://www.accenture.com/us-en/blogs/blogs-reshma-shinde-devops-transformations-operations

As interações entre as áreas de negócio e desenvolvimento são o escopo de ação das abordagens ágeis, o que tem sido endereçado de certa forma. Uma fronteira que até recentemente não possuía apoio era entre as equipes de desenvolvimento e operações.

Particularmente nessa área, o que se percebe são objetivos em princípio diametralmente opostos, ficando a área de desenvolvimento sujeita à cobranças devido às pressões por entregas cada vez mais antecipadas, ao passo que a área de operações é cobrada pela estabilidade dos ambientes.

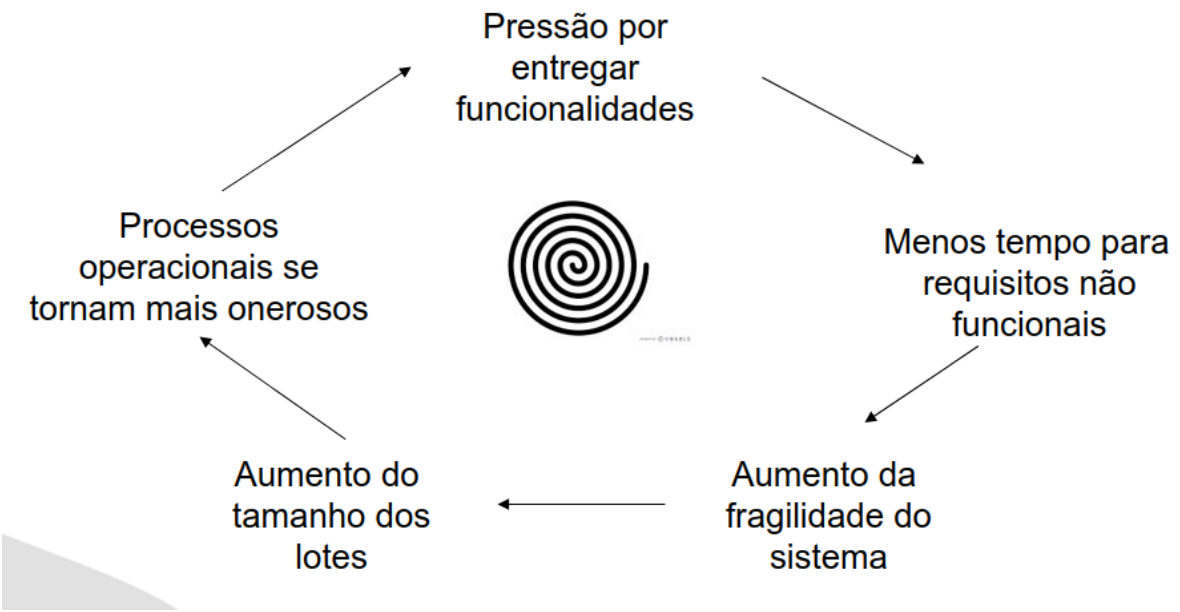
Figura 24 – Muro da Confusão



Fonte: https://www.accenture.com/us-en/blogs/blogs-reshma-shinde-devops-transformations-operations

Essa pressão acaba por gerar o efeito que é a Espiral da Morte: quanto maiores as pressões por entregas antecipadas, maior a negligência pelos aspectos técnicos, o que leva a soluções cada vez mais frágeis, maior dificuldade e esforço em se entregar soluções, o que, por sua vez, aumenta a pressão por novas entregas. Essa espiral acaba alimentando um débito técnico cada vez maior.

Figura 25 –Espiral da Morte



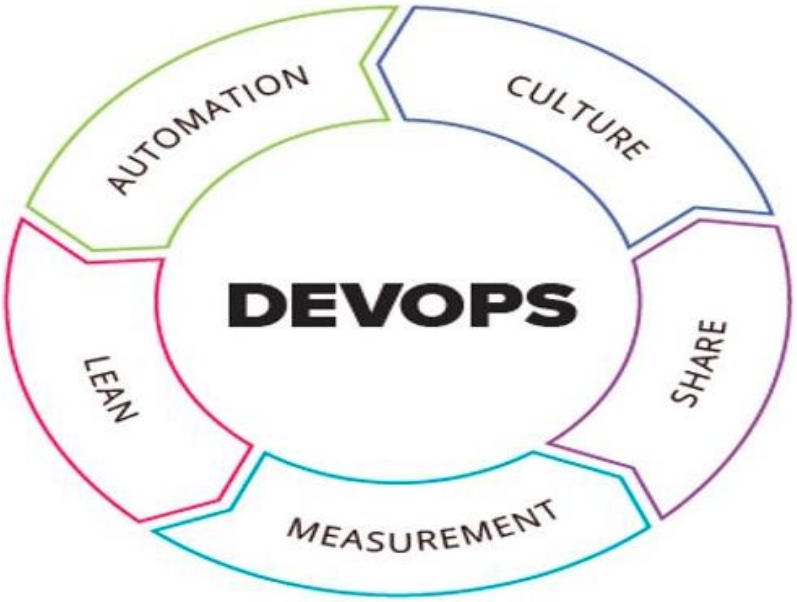
Fonte: Kim, Humble, Willis, Debois.

Visão geral do DevOps

Conforme Gene Kim, uma das maiores autoridades no assunto, o DevOps surgiu, então, como um movimento profissional emergente que preconiza a um relacionamento de trabalho colaborativo entre equipes de Desenvolvimento e Operações, resultando no fluxo rápido do trabalho planejado, com altas taxas de entrega, enquanto de forma simultânea incremental a confiabilidade, estabilidade, resiliência e segurança do ambiente de produção.

DevOps pode ser percebido, ainda, por meio do Modelo CALMS, que envolve aspectos de Cultura, Automação, Fluxo (Lean), Medição e Compartilhamento (Sharing).

Figura 27 – Modelo CALMS



Fonte: https://www.softserveinc.com/en-us/blog/assess-devops-structure-through-calms/

Princípios DevOps

O DevOps possui três princípios (também chamados "Three Ways"): princípios de fluxo, princípios de feedabck e princípios de cultura de aprendizagem e experimentação contínuas.

Princípios de fluxo

Os princípios de fluxo têm como objetivo reduzir o tempo necessário para as mudanças serem disponibilizadas em produção, garantindo a qualidade e confiabilidade das soluções.

Há várias práticas associadas aos princípios de fluxo, dentre as quais podem ser destacadas:

- Criar as fundações do pipeline de deployment;
- Habilitar e praticar integração contínua;
- Habilitar testes automatizados rápidos e confiáveis;
- Definir uma arquitetura para liberações de baixo risco;
- Automatizar e habilitar e liberações de baixo risco.

Princípios de feedback

Os princípios de feedback têm como objetivo criar um sistema de trabalho cada vez mais resiliente e seguro. Exemplos de práticas associadas são a) ver os problemas quando ocorrem; b) criar telemetria para habilitar ver e resolver problemas.

Princípios de cultura de aprendizagem e experimentação contínuas

Os princípios de cultura de aprendizagem e experimentação contínuas têm como objetivo criar uma cultura de alta confiança, reforçando a condição de aprendizes para toda a vida, os *lifelong learners,* que devem assumir riscos no trabalho diário. Exemplos de práticas são: a) institucionalizar a melhoria do trabalho diário; b) habilitar e injetar aprendizagem no trabalho diário.

Referências

BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. *Software Architecture In Practice*. SEI Series on Software Engineering. 3 ed. Boston: Addison Wesley, 2013.

FORD, Neal; PARSONS, Rebecca; KUA, Patrick. *Building Evolutionary Architectures:* Support Constant Change. Newton: O'Reilly, 2017.

HUMBLE, Jez; FARLEY, Davis. *Continuous Delivery***:** Reliable Software Releases through Build, Test, and Deployment Automation. Boston: Addison Wesley, 2010.

KIM, Gene; HUMBLE, Jez; WILLS, John; DEBOIS, Patrick. *The DevOps HandBook.* IT Revolution Press, 2016*.*

MALONE, Michael; ISMAL, Salim; GEEST, Yuri Van*. Organizações Exponenciais: por que Elas São 10 Vezes Melhores, Mais Rápidas e Mais Baratas que a sua.* Alta Books, 2019.

FOWLER, Martin. *Padrões de Arquitetura de Aplicações Corporativas.* Bookman, 2003.

Project Management Institute. *Agile Practice Guide.* 2017.

PRESSMAN, Roger. *Software Engineering – A Practical Approach*. 5 ed. Mc Graw Hill, 2001.

RICHARDS, Mark; FORD, Neal. *Fundamentals of Software Architecture – An Engineering Approach.* Newton: O'Reilly, 2020.

SCHWABER, Ken; SUTHERLAND, Jeff. *O Guia do Scrum*. Disponível em: www.scrum.org. Acesso em: 17 set. 2020.