XPe

Aprenda com quem faz

Fundamentos em Arquitetura de Software

Prof.ª Priscilla Márcia Scarpelli Bastos

2022

SUMÁRIO

> Capítulo 1

Capítulo 1. Introdução à Arquitetura

A arquitetura de software se refere à disciplina de criação de estruturas fundamentais de um sistema de software. No final da década de 60, cientistas, como por exemplo o Edsger Dijkstra, começaram a pesquisar mais sobre o conceito sobre arquitetura de software.

> Uma arquitetura é o conjunto de decisões significativas sobre a organização de um sistema de software, a seleção de elementos estruturais e suas interfaces, juntamente com o comportamento especificado nas colaborações entre estes elementos, a composição destes elementos em subsistemas progressivamente maiores e o estilo arquitetural que guia esta organização. (The Rational Unified Process: An Introduction)

Arquitetura nas organizações

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

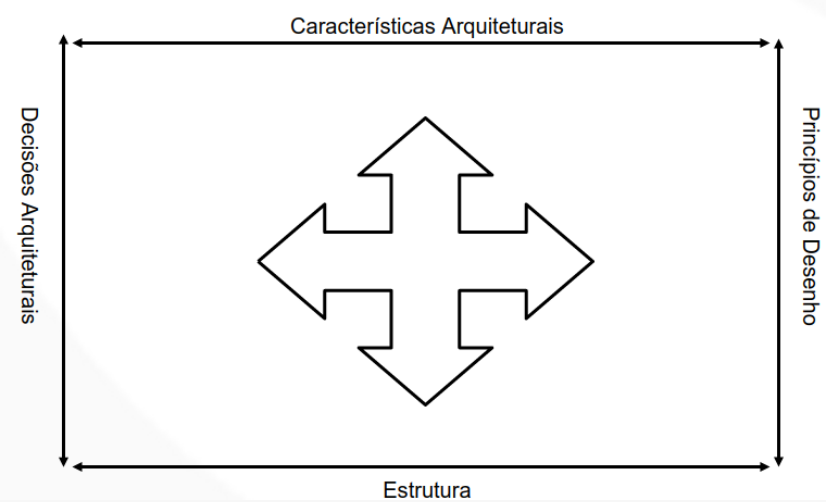
Este primeiro capítulo tem como principal objetivo prover uma visão geral do que é arquitetura de software, as áreas com as quais a arquitetura faz interface e as principais dimensões tratadas nesse assunto.

“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, 2013).

Além disso, a arquitetura também pode ser definida por suas quatro dimensões:

- Características arquiteturais.
- Estrutura.
- Decisões arquiteturais.
- Princípio de desenho.

Figura 1 – Elementos Arquitetura de Software.

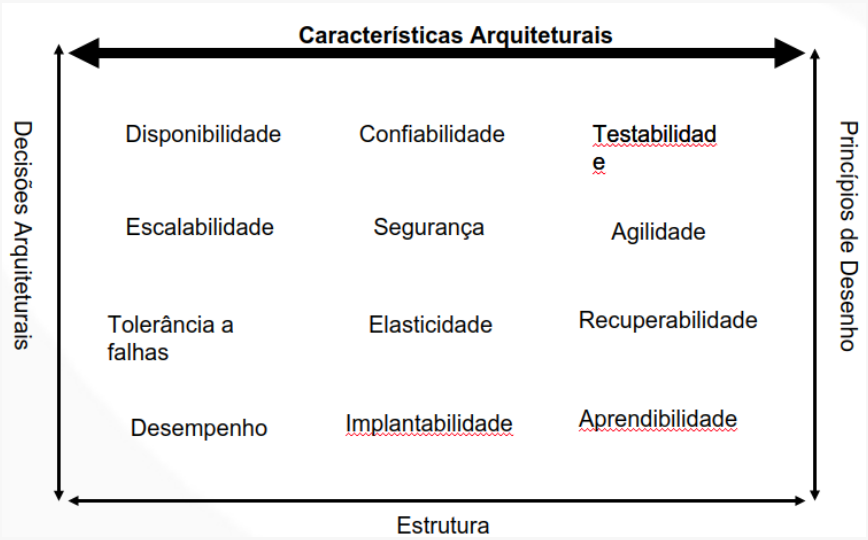


Fonte: Fundamentals of Software Architecture (RICHARDS; FORD, 2020).

Estes elementos podem ser quebrados em características arquiteturais que irá definir os critérios de sucesso do projeto.

Estes elementos podem ser quebrados em características arquiteturais que irá definir os critérios de sucesso do projeto.

Figura 2 – Características Arquitetura de Software.



Trajetória de um arquiteto

O início da trajetória de um arquiteto pode ser fundamentado em três pilares:

Figura 3– Trajetória de um Arquiteto.

Existem alguns bons exemplos que não traçam exatamente, mas podemos dizer que a grande maioria possui esses três fundamentos. Uma boa formação acadêmica na área apoia o profissional com conhecimentos teóricos, que muitas das vezes são importantes para conhecimentos mais

avançados. Alguns cursos de graduação mais comuns dos profissionais de TI são:

- Ciência da Computação.
- Sistema de Informação.
- Engenharias (Software, Computação, Sistemas, Controle e Automação, Elétrica, dentre outras).
- Além da graduação, muitas vezes o arquiteto também possui alguma pós-graduação como, por exemplo:
- Arquitetura de Software.
- Engenharia de Software.

Mas fora as formações também é importante que o profissional tenha algumas especializações para conhecimentos em tecnologias e ferramentas do mercado. Essas especializações podem ser em tecnologias, ferramentas ou em conhecimentos gerais, por exemplo:

- Tecnologia:
  - Linguagens (Java, Python, C#).
  - Banco de Dados (Cosmos, SQL, MySQL).
  - Framework frontend (React, Angular, Vue).
  - Framework backend (Django, Laravel, .Net, Spring Boot).
- Ferramentas:
  - IDE's de desenvolvimento (VS Code, Visual Studio, Eclipse).
  - Gerenciamento de Versão (Git, SVN, TFS).

- Ferramentas de Modelagem (UML, ArchiMate).
- Ferramentas de Apoio.

- Conhecimentos:
  - Orientação a objetos.
  - Designer Patterns.
  - Integração Contínua.
  - Gestão de Pessoas.
  - Negociação e Comunicação

Um outro ponto muito fundamental para o arquiteto de softwares é a experiência prática, não só de conhecimentos teóricas que se dá a trajetória de um arquiteto. É importante que o arquiteto tenha experiências para poder tomar as melhores decisões em momentos decisivos e em momentos de crise.

Um outro ponto muito fundamental para o arquiteto de softwares é a experiência prática, não só de conhecimentos teóricas que se dá a trajetória de um arquiteto. É importante que o arquiteto tenha experiências para poder tomar as melhores decisões em momentos decisivos e em momentos de crise.

> Capítulo 2

Capítulo 2. Arquitetura de Software X Soluções X Corporativa

No que tange a disciplina de arquitetura no desenvolvimento de um software, nos deparamos com vários tipos de arquitetura. Destes tipos temos três que podemos citar, que são a arquitetura do software, arquitetura de soluções e arquitetura corporativa. Todos estes tipos de arquitetura acabam fazendo interface umas com as outras em algum momento durante o desenvolvimento, mas algumas atividades podem ser agrupadas sob a governança de cada uma destas áreas.

Arquitetura de Software

Esta área é a responsável pela organização da solução em um nível mais próximo do código e dos desenvolvedores. Algumas decisões que são tomadas por essa disciplina são, por exemplo:

- Qual stack e linguagem será utilizada. Por exemplo, Java ou .Net, Python.
- Quais frameworks e bibliotecas que serão utilizadas na solução. Por exemplo, em uma aplicação em .Net, deve-se tomar a decisão se será utilizado o Automapper para fazer o mapeamento entre os objetos ou se isso será feito manualmente.
- Como será o acesso à camada de dados, ou seja, será utilizado um ORM ou será utilizado comandos com as queries diretas na camada de dados, sem o intermédio de um framework.
- Qual o modelo arquitetural que utilizaremos na implementação, como MVC, em camadas, hexagonal, onion etc.

É importante que o arquiteto de softwares tenha um bom conhecimento técnico, o mais próximo de um desenvolvedor sênior, para que consiga tomar estas decisões de modo que o software seja robusto na

medida do necessário e para que o desenvolvimento seja fluido e cadenciado.

O arquiteto de softwares deverá estar preparado para agir em momentos de crise. Muitas vezes será demandado a ele que seja capaz de identificar soluções para problemas específicos, que dificilmente serão identificados por desenvolvedores menos capacitados. Algumas vezes poderá ser necessário realizar debugging até mesmo no mais baixo nível dos frameworks para tomar uma decisão, que deverá corrigir o problema pertinente e fazer com que o negócio volte a conseguir operar o sistema que está parado em crise.

Em vários momentos, o arquiteto de softwares precisará fazer interface com o negócio, mesmo que isso seja mais inerente ao arquiteto de soluções. Com isso é necessário que o mesmo tenha uma boa habilidade de comunicação para conseguir trabalhar o requisito de negócio e conseguir traduzi-lo para padrões arquiteturais e de implementação, para que a equipe de desenvolvimento consiga executar.

Uma etapa importante do desenvolvimento, que não é mandatória, mas é importante que o arquiteto de softwares consiga participar, é durante as revisões de código. Neste momento o arquiteto conseguirá validar se os padrões e boas práticas que foram determinados no início do projeto estão sendo devidamente seguidos.

A arquitetura de software deverá sempre estar alinhada a boas práticas de desenvolvimento que são divulgadas no mercado e ter senso crítico sobre estas práticas para aplicá-las à medida que fizer sentido para os softwares a serem desenvolvidos. Cabe aos arquitetos de software também evangelizarem os desenvolvedores quanto a estas boas práticas.

Arquitetura de Soluções

A arquitetura de soluções está em uma camada superior à arquitetura de software. Esta disciplina é responsável por transformar o requisito de negócio em componentes de engenharia que irão atuar em conjunto para resolver o problema de negócio.

A principal diferença entre a função de arquiteto de soluções e a função de arquiteto de softwares é que o último trata apenas de questões de engenharia. Por outro lado, um arquiteto de soluções é responsável por garantir que um produto de software resolva um problema de negócios específico dentro da estratégia de uma empresa.

Algumas das atividades estão associadas a esta disciplina são:

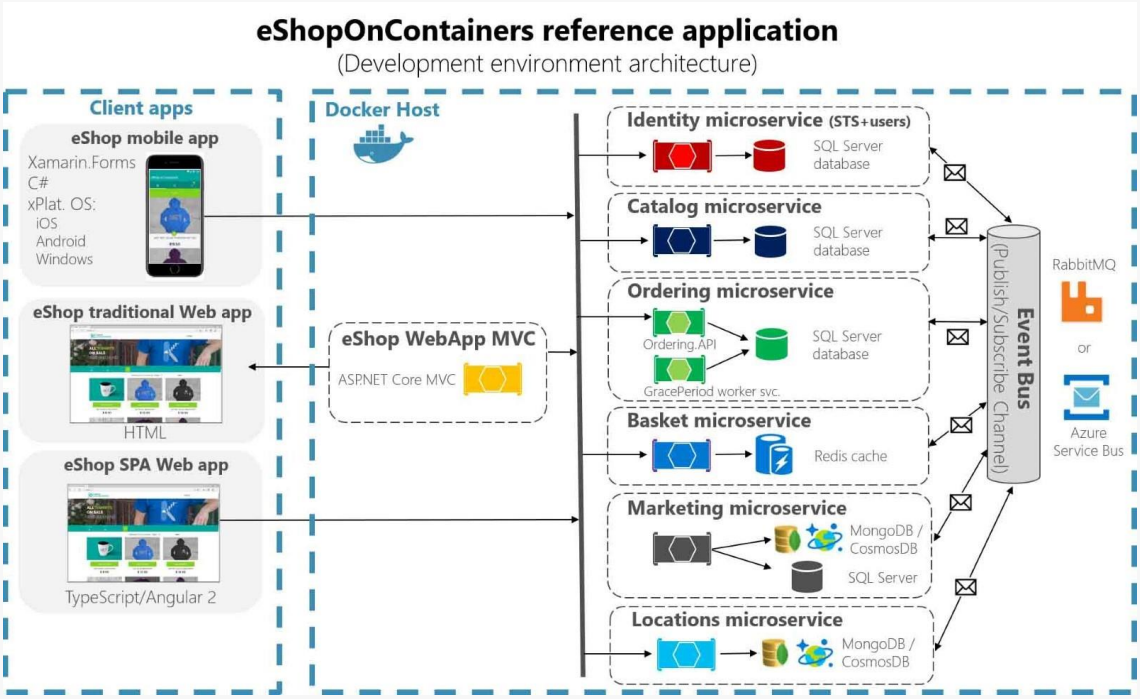
- Criar e liderar os processos de integração entre sistemas para atender aos requisitos de negócio.
- Avaliar as restrições do projeto para encontrar alternativas e avaliar os riscos intrínsecos à tecnologia.
- Atualizar as partes interessadas sobre o andamento do desenvolvimento de um software e os custos.
- Notificar as partes interessadas sobre quaisquer problemas relacionados à arquitetura.
- Analisar o impacto comercial que certas escolhas técnicas podem ter nos processos de negócios.
- Estar alinhado à estratégia tecnológica da organização.
- Analisar os requisitos não funcionais que permeiam a solução (tempo de resposta, volume de acesso etc.).

Podemos ver que o arquiteto de soluções está na divisão entre técnico e negócio e o mesmo deve *"jogar"* dos dois lados, uma vez que é fundamental que a solução técnica que está sendo definida resolva um problema de negócio.

Mais ainda que o arquiteto de softwares, as habilidades de comunicação são exigidas nesta disciplina, sendo que muitas vezes, o arquiteto de soluções estará em comunicação direta com o negócio e que, normalmente, tem o conhecimento técnico muito pequeno ou inexistente. Com isso, o arquiteto de soluções deve ser capaz de explicar de forma simples para o negócio todo a solução técnica que será construída, de uma forma abstrata o suficiente para que todos entendam.

Nestas habilidades de comunicação, algo que é muito importante é saber desenhar a solução através de algum diagrama. Existem várias formas, como utilizando notações BPMN ou C4, e várias ferramentas como draw.io e Archimate, que servem para facilitar este tipo de desenho. O mais importante é que o diagrama represente de forma clara os componentes do software, a interface com outros sistemas, o fluxo de dados e mais alguma informação que for pertinente tanto para a engenharia quanto para o negócio. A figura a seguir é um exemplo de como fazer um diagrama arquitetural de um sistema.

Figura 4 – Exemplo de diagrama arquitetural. Solução eShop on Containers utilizada pela Microsoft para divulgar boas práticas de .Net.



Fonte: https://github.com/dotnet-architecture/eShopOnContainers

As decisões arquiteturais tomadas para a definição da solução é algo que deve ser documentado e formalizado, então cabe ao arquiteto de soluções tomar nota disto e divulgar nos canais da organização em que outros arquitetos tenham acesso de modo a recapitularem e contribuírem com as decisões tomadas. Esta documentação ainda serve de fonte para que outros arquitetos consigam iniciar outras soluções tomando como base a experiência dos outros.

É muito comum que algumas decisões tomadas no início da concepção de um produto/software não se mantenham quando este vai crescendo ao longo do tempo. Isso pode acontecer quando algum componente definido no início não suporta a escala necessária e precisa ser substituído por um mais robusto, com isso, cabe ao arquiteto de soluções comunicar a todos os interessados do software que esta ação será necessária e apresentar todos os impactos, sejam financeiros, de tempo ou quaisquer outros impactos.

Arquitetura Corporativa

A arquitetura corporativa pode ser definida como uma série de conhecimentos técnicos e habilidades de gestão que integram a tecnologia aos objetivos do negócio. Neste ponto pensamos na arquitetura de uma forma mais abrangente, como algo que permeia estrategicamente a organização como um todo. É a organização lógica para processos de negócio e infraestrutura de TI, refletindo a integração e a padronização para o modelo operacional da organização.

Neste ponto, é papel da arquitetura corporativa definir métodos e ferramentas que irão atender a organização como um todo. Estas definições deverão se tornar padrões a serem seguidos por todas as equipes naquela organização que estão desenvolvendo e entregando software.

Algumas decisões que são tomadas a nível de arquitetura corporativa podem ser:

- Padronização de Linguagem e frameworks a serem utilizados nas soluções. Ex.: é decidido que toda a organização irá utilizar .Net na versão 5 com C# para aplicações de backend.
- Padronização das bases de dados. Ex.: todos os squads utilizarão o Microsoft Sql Server como base de dados relacional.
- Processos de controle de mudança para auditoria de publicações em ambiente produtivo.

É muito comum em organizações que ainda não adotaram a TI como área estratégica que não exista estes padrões bem definidos, por isso cabe à arquitetura corporativa realizar atividades como:

- Documentar o modelo arquitetural atual que a organização está trabalhando, por mais que não esteja totalmente organizado, ele vai ajudar a definir o novo modelo.

- Definir e documentar um modelo de arquitetura o qual a organização vai trabalhar para seguir no futuro.
- Definir um plano de migração do modelo antigo para o modelo novo.
- Avaliar a implantação de novas tecnologias.
- Monitorar o uso das tecnologias e coletar feedback dos desenvolvedores para implantar novas melhorias.
- Implementar soluções corporativas de DEVOPS.
- Criar comitês técnicos.
- Avaliar custos das soluções implementadas.

Estudos de Caso

Restrições de Projeto

Você é um arquiteto de soluções que trabalha na Mobile Max, uma empresa de telefonia celular que vende planos para pessoa física no Brasil. A área de vendas consegue firmar uma parceria com a Total Security, uma empresa que vende apólices de seguro diversos. Esta parceria consiste em os clientes Total Security que contratarem um plano de telefonia na Mobile Max teriam um bônus no consumo de internet de mais 5gb. A área de negócio das duas empresas quer fazer o MVP desta nova funcionalidade e lançar em até 1 mês, divulgando para o mercado. A Total Security usualmente já faz essas integrações com outros parceiros e disponibiliza uma Api Rest autenticada, que recebe o documento do cliente e retorna indicando se o cliente é ou não cadastrado. A área de negócio da Mobile Max não pretende investir muito dinheiro nesta integração no primeiro momento, pois quer avaliar se efetivamente será uma boa parceria. Você deve definir o modelo arquitetural de comunicação baseado nas restrições definidas para o MVP, assim como indicar os trade-offs das decisões tomadas. Todos os planos são criados através de um serviço centralizado

legado, que fica hospedado na infra interna da Mobile Max, e esse serviço é consumido pelos mais diversos canais de atendimento (site, central de atendimento, aplicativo de celular).

O case descrito acima é um clássico que o negócio quer a implementação mais simples e mais rápida para validar a viabilidade de um produto no mercado. Foram colocadas as seguintes restrições:

- Tempo, temos 1 mês para fazer a implementação, testar e colocar em produção.
- Estamos restritos a utilizar a API REST disponibilizada pela Total Security.
- Devemos utilizar o mínimo de recurso financeiro para isso.

A alternativa mais simples que temos de implementação neste cenário é:

1. Fazer uma alteração no sistema centralizado de gestão de planos para que a cada novo plano criado, será feita uma chamada à api da Total Security para verificar se o cliente é um segurado.
    a. Se positivo, adicionamos mais 5gb de limite de internet no plano do cliente;
    b. Caso negativo não adicionamos.

Com isso conseguimos entregar a feature em produção e vai atender como um MVP, mas criamos alguns trade-offs:

- Inserimos uma dependência externa dentro do fluxo de venda de planos, com isso, se não for bem implementado, instabilidades criadas na api da Total Security podem impactar diretamente a ativação de novos planos de telefonia.

- Muito provável que a maioria das ativações de planos não serão de clientes da Total Security, com isso todos estes clientes seriam impactados com um tempo maior na ativação.

- Precisaremos fazer uma alteração em um sistema legado, algo que é sempre arriscado para a organização.

Como vimos acima, atendemos ao requisito de negócio da organização, mas temos os trad-offs que o arquiteto de soluções deverá compartilhar com o negócio para que todos estejam cientes dos riscos envolvidos.

Cabe ao arquiteto de soluções também fazer um desenho para a solução definitiva do problema e determinar o esforço necessário para ser executado. Como solução definitiva para este produto, poderíamos ter uma aplicação batch que executa periodicamente, verificando todos os planos ativados no intervalo das execuções e consultar os clientes na api da Total Security e adicionar os 5gb de limite de internet somente para os planos em que retornaram positivo. Essa solução resolve nossos três trade-offs:

- Eliminamos a dependência no fluxo de ativação de planos.

- Não impactamos os clientes que não são clientes da Total Security.

- Não precisamos alterar o sistema legado.

Mas ainda temos trade-offs:

- Criamos mais um artefato para ser gerenciado, com isso precisamos de mais monitoramento, mais recurso de hardware ou até mais um pipeline de publicação.

- Falhas nesse processo podem ficar invisíveis ao negócio, pois executará de forma assíncrona e, se parar de executar, possivelmente saberemos que está com problema só quando o

cliente entrar em contato questionando porque não recebeu o bônus.

Podemos evoluir ainda mais essa solução até ampliando o escopo para uma migração da arquitetura do legado para trabalhar com eventos por exemplo e isso traria ainda mais trade-offs. O importante aqui é mostrar que sempre que o arquiteto de soluções for propor uma solução ele deverá lidar com restrições e deverá demonstrar os trade-offs das decisões tomadas para as partes interessadas.

Proposta de Arquitetura Corporativa

Nesta sessão vamos elaborar bem em alto nível uma proposta arquitetural corporativa para uma organização de médio porte e algumas questões que devem ser discutidas.

Nosso estudo de caso será sobre a Acme Equipamentos, uma empresa especializada em equipamentos e ferramentas para construção civil, trabalhando com aluguel, manutenção e venda. A empresa foi fundada em 2001 e hoje conta com mais de 2000 funcionários e 50 filiais em todo Sudeste. Todo aparato técnico é baseado em softwares de prateleira, o que não está permitindo a escalada de novas funcionalidades. A empresa já possui um departamento de TI com poucos desenvolvedores e com a maior parte dos integrantes sendo pessoas de infraestrutura, que são encarregadas de sustentar e manter o funcionamento dos servidores de banco de dados e de aplicação. A diretoria deseja contratar mais profissionais, principalmente desenvolvedores, para o departamento de TI. Como arquitetos corporativos, fomos demandados de definir uma estrutura básica que será utilizada pelos desenvolvedores para entregar as aplicações em ambiente de produção.

Vamos listar agora algumas decisões que podemos tomar para esta estruturação:

- Todas nossas aplicações serão entregues no formato web. Sendo o frontend desenvolvido utilizando ReactJS com typescript e o backend desenvolvido utilizando .Net 6 no formato de Api REST. Todas estas aplicações deverão ser encapsuladas dentro de containers Docker.

- As aplicações serão publicadas utilizando a nuvem Azure. Os componentes utilizados por padrão serão Azure Web App, publicado com containers.

- Novas aplicações deverão criar um banco de dados individual. Caso seja necessário um banco NoSql, deverá ser utilizado o CosmosDb e, quando necessário um banco relacional, deverá ser utilizado o Azure Sql.

- Instalaremos uma estrutura de Change Data Capture nos bancos de dados dos sistemas legados e as alterações executadas de dados nestas databases serão publicadas no Azure Event Grid. Novas aplicações poderão ouvir estes eventos conectando diretamente no Event Grid.

- Quando necessária a sincronização de informações dos sistemas novos com os sistemas antigos, deverá ser publicado no ambiente on-premise uma api que funcionará como Camada Anticorrupção e estes dados deverão ser sincronizados através desta api sempre que possível de forma assíncrona.

Acima temos uma fração bem pequena de decisões que possivelmente precisaremos tomar definindo uma arquitetura corporativa para uma empresa. O mais importante aqui é que estas decisões se tornem padrões, para facilitar o trabalho e a troca de recursos entre squads e para contribuir com a produtividade dos desenvolvedores.

> Capítulo 3

Capítulo 3. Hard e Soft Skills

A palavra “skills” do inglês pode ser traduzida como competências ou habilidades. Então nessa seção iremos abordar um pouco mais afundo as skills já apresentadas no capítulo anterior e algumas outras que são importantes para um arquiteto. Dessa forma, iremos dividir essas habilidades em *Hard* e *Soft skills,* onde as hard skills são as habilidades técnicas e as soft skills são as comportamentais.

Hard Skill

As Hards Skills são as habilidades técnicas adquiridas na sala de aula, durante alguma formação específica ou pela experiência. Chamamos de hard skills as habilidades profissionais que são quantificáveis, ou seja, que podem ser mensuradas de alguma maneira. Por conta disso, essas competências podem ser comprovadas por meio de certificações e diplomas, por exemplo. Alguns exemplos de hard skills são:

1. Programação Orientada a Objetos.
2. SOLID.
3. Design Patterns.
4. DDD.
5. CQRS.
6. Clean Code.

Soft Skill

1. Criatividade:

Responsável por transformar requisitos de negócio em requisitos técnicos, o arquiteto deve ser capaz de entender e desenhar a melhor

maneira de se resolver um problema e pensar em como todo o arcabouço tecnológico pode ser utilizado para poder resolver o problema de forma mais robusta, barata e rápida, respeitando as premissas envolvidas.

2. Senso Crítico.

3. Proatividade.

Como o arquiteto é responsável por ajudar identificar e gerenciar riscos, monitorar a saúde da aplicação, garantir qualidade, dentre outras funções, a soft skill de proatividade é de extrema importância para que se identifique problemas o quanto antes e defina planos de ações antes que o problema fique crítico.

4. Resiliência.

5. Comunicação.

O Arquiteto deve trabalhar em colaboração com outros profissionais de TI, por isso deve ter habilidades colaborativas mais amplas. Deve ter a capacidade de trabalhar bem com indivíduos e departamentos de toda a organização. Essa é uma das razões pelas quais as habilidades de comunicação são tão críticas. Isso requer:

- Capacidade de expressar conceitos complexos verbalmente e por escrito, tanto para o público técnico quanto para não técnicos.
- Habilidades de apresentação.
- Capacidade de ouvir “atentamente”.

> Capítulo 4

Capítulo 4. Papel do Arquiteto

Nos capítulos anteriores vimos um pouco sobre os tipos de arquitetura que existem (Software, Solução e Corporativa) e vimos um pouco sobre as Hard e Soft skills necessárias para exercer o papel de um arquiteto. Mas afinal, o que seria esse papel de arquiteto? Quais são as funções do arquiteto? Ele deve ser programador e ter conhecimento de desenvolvimento? Ele é responsável por tomar todas as decisões sozinho?

O que seria o Papel do Arquiteto?

A arquitetura é o primeiro passo do desenvolvimento de um projeto de software. Nele, o arquiteto é responsável por ajudar a conectar as necessidades das áreas de negócio com a solução técnica. Este profissional é responsável por garantir algumas diretivas de um projeto como, por exemplo, qualidade da solução, performance, usabilidade, baixo custo, desempenho, funcionalidades, contexto organizacional, liderança e gestão de pessoas. São eles os responsáveis a dar suporte na tomada de decisão de alto nível, ajudando na decisão de medidas técnicas, padrões de programação, designer e frameworks.

Mas os arquitetos precisam tomar todas as decisões por conta própria e definir todas as diretivas sozinho? Não, uma tarefa importante do arquiteto é se colocar na posição de liderança técnica. Nessa posição ele deve expor a sua experiência, mas a medida do possível deve oferecer a liberdade para a sua equipe participar das decisões arquiteturais, possibilitando que todos participem das definições das tecnologias, frameworks e matrizes de decisões. Essa gestão do conhecimento sem engessar o software e centralizar as decisões possibilitam o desenvolvimento de um time mais motivado, que participa, sabatina e agrega com experiências e conhecimentos adquiridos em outros contextos.

Devemos lembrar que as decisões arquiteturais tomadas deverão acompanhar o projeto durante o ciclo de desenvolvimento.

A arquitetura de um sistema é o planejamento da construção das classes de softwares, que faz o intermédio entre o hardware e as aplicações, de maneira que o arquiteto precisa conhecer quais são as tecnologias disponíveis e onde elas se encaixam melhor. Mas esse papel tem ainda uma dificuldade a mais, a Engenharia de Software, consequentemente a Arquitetura de Software, é hoje uma área que sofre grande mudanças e evoluções tecnológicas e de paradigmas, exigindo constantes atualizações dos seus profissionais. Por exemplo, na década de 70 o conceito de não replicar uma informação era muito divulgado, mas hoje muitas soluções, devido as premissas e restrições, podem ser inviáveis manter esse paradigma. Se considerarmos um cenário onde precisamos buscar um volume de dados muito grande com uma performance muito alta o relacionamento de tabelas para complementar a informação, pode ser muito custosa e não atender ao requisito de performance, podendo se ver necessário a réplica de algumas informações e uma estruturação dos dados de forma diferenciada (como no caso dos bancos NoSQL, cache e dentre outros recursos que precisam de escalabilidade horizontal).

Atividades de um Arquiteto

Então como podemos ver que as atividades do arquiteto passam tanto pelas atividades técnicas quanto algumas atividades de comunicação e gestão. Podemos enumerar essas atividades como:

1. Identificar os stakeholders:

Os stakeholders são indivíduos, times, organizações que apresentam interesse na realização do projeto. Conhecê-los é essencial para a geração de valor, são eles que irão trazer para a arquitetura do projeto as necessidades, premissas e restrições. Cada stakeholder exerce um grau de

influência diferente na arquitetura do projeto, por isso além de identificá-los é importante classificá-los para garantir a devida atenção a cada um deles.

Uma boa arquitetura endereça bem as preocupações dos stakeholders e em caso de conflito consegue equilibrá-las através de uma análise de impacto para garantir que a solução atenderá a todos de forma aceitável. A arquitetura de um software pode ser tratada como um modelo relativamente pequeno e compreensível de como o sistema será organizado e como os componentes irão trabalhar em conjunto. Esse modelo deve ser muito utilizado para que a maioria dos stakeholders consigam ter um entendimento base e a partir dele consigam se comunicar e caso necessário consigam ter negociações e chegarem em consenso sobre as melhores decisões. Alguns stakeholders no processo de definição de uma arquitetura podem ser:

- Analista de Requisitos / Product Owner / Usuário / Designer: Serão responsáveis por trazer para o projeto as dores da área. Será a partir dessas dores que serão definidas as necessidades e funcionalidades do projeto. São esses stakeholders que trazem para o projeto o que a solução precisará ter para gerar valor.

- Tester / Analista de Segurança / Equipe da Infra: Serão responsáveis por trazer algumas restrições e boas práticas para o projeto. Muitas das vezes eles poderão trazer um direcionamento sobre tecnologias homologadas e utilizadas na organização. Diretivas e tecnologias importantes de serem seguidas e utilizadas para garantir a segurança das informações dentro de todos os fluxos da solução.

- Desenvolvedores / Equipe de suporte: Serão responsáveis pela implementação dos componentes e suporte dos mesmo em produção. Podem trazer restrições com relações a alguma tecnologia ou a premissa de precisarem de treinamento e formação, dependendo do que for escolhido para a arquitetura.

- Gerente de projeto / Scrum Master: poderão trazer para o projeto restrições de custo e/ou de prazo. É sempre importante manter uma comunicação com esse stakeholder para que as expectativas estejam alinhadas.

- Outros: Além dos perfis de stakeholders apresentados, pode ser necessário a comunicação com outros stakeholder técnicos, gerenciais ou da área de negócio. Por isso a comunicação é sempre uma ferramenta muito importante.

2. Análise de trade-off e visibilidade:

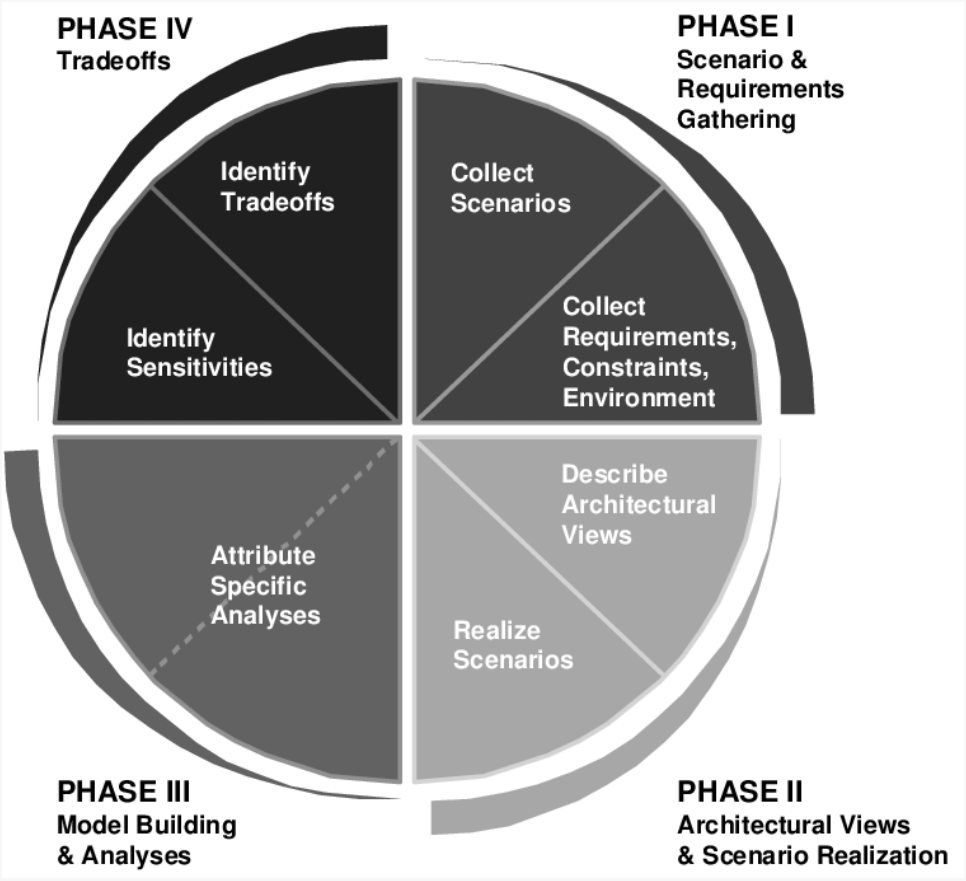
Análise de trade-off é algo extremamente comum em software. Basicamente, a expressão em inglês significa o ato de escolher alguma coisa em detrimento de outra e vai consistir em uma análise de pontos positivos e negativos de cada uma das escolhas que faremos. Isto significa basicamente que escolhendo uma opção estaremos carregando os pontos negativos e positivos dela e estaremos perdendo os pontos positivos e negativos da outra opção.

Vamos para o seguinte exemplo, ao guardar dados em memória, podemos ter ganhos de performance associados ao acesso rápido, porém teremos maior consumo de memória pela aplicação e o dado pode estar desatualizado, pois a fonte original não é consultada diretamente, ao passo que, consultar diretamente a fonte de dados original, teremos uma maior confiança que o dado está correto, porém lidaremos diretamente com o custo de performance para acessá-lo.

Para nós o objetivo de uma análise de Trade-Off será nos ajudar a escolher uma arquitetura e tecnologias agregadas, que serão adequadas para o problema de negócio que iremos solucionar com esse software que estamos produzindo. Na engenharia de software existe um método que ajuda nesse processo de decisão, o ATAM (Architecture Tradeoff Analysis

Method). Seu objetivo é ajudar a escolher uma arquitetura adequada para um sistema de software, descobrindo compensações e pontos de sensibilidade.

Figura 5 – Análise de Trade-off.



Fonte: https://www.researchgate.net/.

- Apresentar ATAM – Apresentar o conceito de ATAM aos stakeholders e esclarecer dúvidas sobre o processo.
- Representantes da área de negócio – todos no processo apresentam e avaliam os motivadores de negócios para o sistema em questão.
- Apresente a arquitetura - o arquiteto apresenta a arquitetura de alto nível para a equipe, com um 'nível de detalhe apropriado'

- Identifique as abordagens arquitetônicas – diferentes abordagens arquitetônicas do sistema são apresentadas pela equipe e discutidas.

- Gerar árvore de utilitários de atributos de qualidade - definir o negócio principal e os requisitos técnicos do sistema e mapeá-los para uma propriedade arquitetônica apropriada. Apresente um cenário para este determinado requisito.

- Análise das abordagens arquitetônicas – analise cada cenário, classificando-os por prioridade. A arquitetura é então avaliada em relação a cada cenário.

- Faça um brainstorming e priorize cenários – entre o grupo maior de partes interessadas, apresente os cenários atuais e expanda.

- Analise as abordagens arquitetônicas – execute a etapa 6 novamente com o conhecimento agregado da comunidade mais ampla de partes interessadas.

- Apresentar resultados – fornece toda a documentação às partes interessadas.

Durante todo o processo do ATAM, as soluções encontradas serão sabatinadas e analisada a viabilidade, até que se defina o modelo arquitetural do projeto.

3. Modelagem arquitetural:

A modelagem arquitetural é uma ferramenta muito poderosa para ajudar na compreensão da solução por todos os stakeholders do projeto. De acordo com Booch, Rumbaugh e Jacobson, há quatro objetivos principais para se criar modelos:

- Eles ajudam a visualizar o sistema como ele é ou como desejamos que ele seja.
- Eles permitem especificar a estrutura ou o comportamento de um sistema.
- Eles proporcionam um guia para a construção do sistema.
- Eles documentam as decisões tomadas no projeto.

A escolha de um modelo, dado os mesmos requisitos de software, pode ser completamente diferente de acordo com a experiência da equipe de projetista envolvida. Além disso, é importante ter em mente que nenhum modelo único é suficiente, para ser melhor definido e investigado é importante termos um conjunto de modelos independentes que apresente a solução por diferentes óticas. Cada uma dessas diferentes visões conterá aspectos comportamentais e estruturais, sendo que o conjunto representará a base do projeto de software.

Para ajudar na compreensão da modelagem o arquiteto poderá utilizar do artefato de diagramas para poder melhor representar a estrutura do projeto. Para permitir que todos envolvidos no projeto possam entender mais facilmente os produtos representados em um diagrama, foi definido uma linguagem padrão para se seguir, que é denominada de UML (Unified Modeling Language). Nos próximos capítulos iremos aprofundar um pouco mais nos conceitos e regras definidas na UML.

4. Prototipação, simulação e realização de experimentos:

Uma outra atividade de extrema importância para o arquiteto é a de realizar alguns testes de validação da solução proposta, as chamadas POCs (Proof Of Concept). Esses testes são a evidência documentada de que um software pode ser bem-sucedido. Ao desenvolver uma POC, é possível antecipar problemas técnicos que poderão interferir no funcionamento

esperados. Além disso, a prova de conceito permite a solicitação de feedbacks internos e externos. É importante quando se está planejando uma POC considerar:

- Defina as necessidades e objetivos estratégicos.
- Torne a prova mais transparente possível.
- Abra um canal de comunicação direta com os stakeholders durante a evolução da POC.
- Esclareça todas as expectativas e objetivos da POC.
- Estabeleça um comitê de avaliação.

5. Liderança técnica:

A atividade de liderança e referência técnica é muito importante no papel do arquiteto. Algumas tarefas importantes no dia a dia do arquiteto são puxar:

- Sabatina técnica: junto do squad se reunir, inclusive com membros de outros squads, para discutir e sabatinar uma determinada solução técnica que tiver uma complexidade maior e necessitar de um pensamento mais *"fora da caixa"* e ao mesmo tempo utilizar as experiências de outros profissionais para contribuir na formulação da solução técnica.
- Especificação técnica: algumas funcionalidades a serem desenvolvidas, principalmente envolvendo integrações com outros sistemas. Vale a pena escrever uma documentação técnica, além da de negócio, que vai auxiliar na implementação dentro do squad e também vai servir de referência para outras pessoas no futuro.

- Padrões de desenvolvimento: Definir, acompanhar a execução e dar suporte caso aja alguma dúvida da equipe de desenvolvimento quanto a execução dos padrões.

6. Acompanhar o ciclo de desenvolvimento

Boas práticas de desenvolvimento são de responsabilidade de toda a equipe de desenvolvimento, mas, como liderança técnica da equipe, os arquitetos têm uma responsabilidade maior na evangelização destas boas práticas. Abaixo temos algumas boas práticas de desenvolvimento que podem ser seguidas e acompanhadas pelo arquiteto de softwares:

- Revisão de código: toda atividade codificada pelos desenvolvedores deveria passar por uma revisão por outro membro do time, sendo que este membro que faz a revisão pode ser o arquiteto de softwares ou não. A revisão de código ajuda a garantir que os padrões de desenvolvimento pré-determinados no início do desenvolvimento estão sendo completamente seguidos.

- Teste em dupla: para algumas funcionalidades, pode caber ao arquiteto fazer testes em dupla com os desenvolvedores para validar as funcionalidades mais críticas que estão sendo desenvolvidas.

- Análise estática de código: utilizar ferramentas de análise estática de código, a mais utilizada hoje é o SonarQube, mas cada linguagem ainda possui o seu linter que funciona de complemento ao SonarQube.

- Testes automatizados: na conjuntura atual, é imprescindível o desenvolvimento de testes automatizados nas aplicações. Estes testes tem o papel de garantir uma segurança na implementação de novas features de modo que as que foram entregues anteriormente não sejam impactadas.

Várias outras práticas podem ser incluídas aqui, inclusive muitas empresas possuem práticas próprias que são adequadas a cada cenário de cada projeto específico.

7. Monitorar a saúde da aplicação

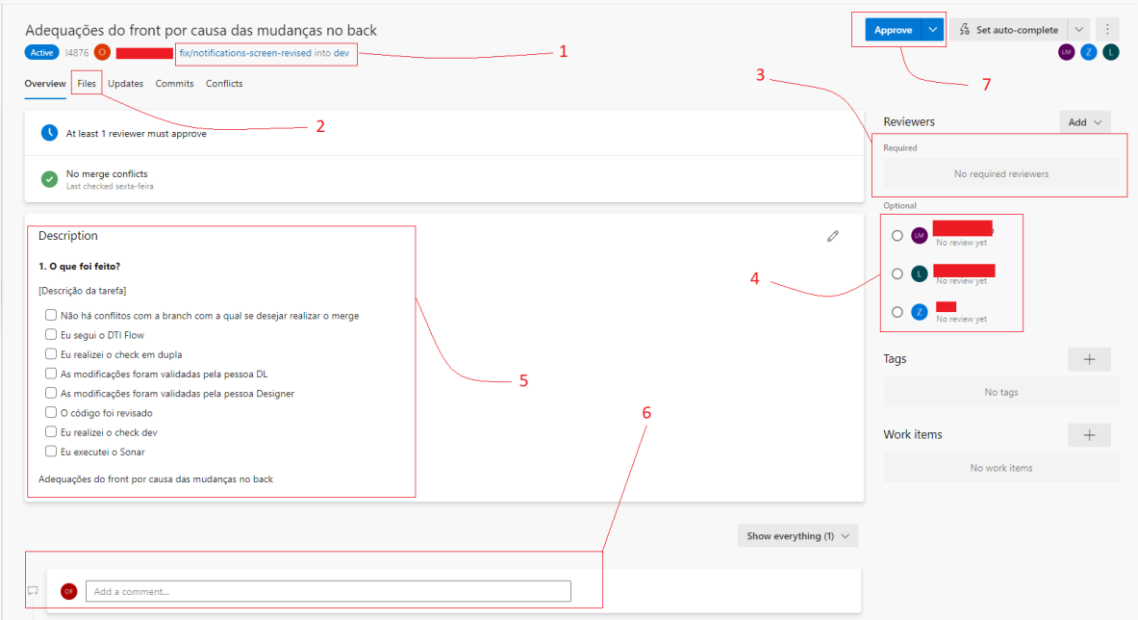
Estudo de Caso

Revisão de Código e Pull Request

Como arquitetos estaremos o tempo todo propondo formas de implementar determinados requisitos de negócio e cabe a nós também apoiar os desenvolvedores nessa forma de implementação.

Abaixo temos um exemplo de como é a interface para aprovação de um pull request utilizando o Azure DevOps.

Figura 6 – Pull Request Azure DevOps.



1. Identificamos o branch de origem e o de destino do pull request.
2. Podemos ver os arquivos que estão sendo alterados neste pull request.

3. Aqui exibiremos uma lista dos aprovadores que são obrigatórios neste pull request, no exemplo não temos nenhum.

4. Aqui temos uma lista de aprovadores opcionais.

5. Temos a descrição do pull request, é importante que se tenha uma descrição clara do que está sendo entregue, aqui ainda temos uma funcionalidade adicional que nos permite criar checklists no Azure Devops.

6. Um espaço para colocarmos comentários para a pessoa que está criando o pull request.

7. É o botão onde aprovaremos ou reprovaremos o pull request.

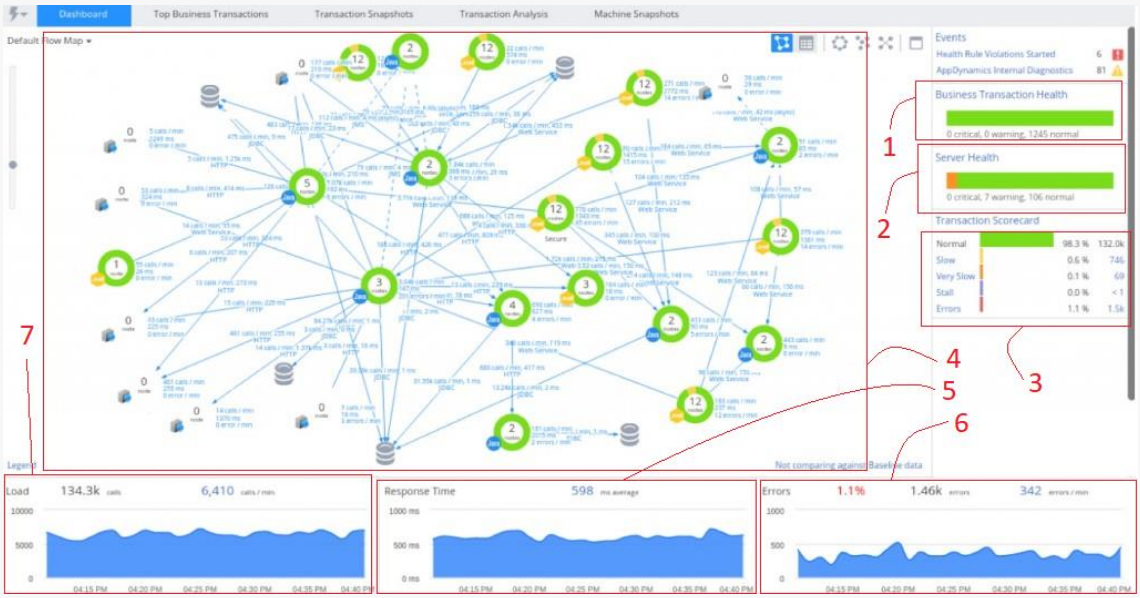
O Azure devops é uma ferramenta muito poderosa para a gestão de um código e estes processos auxiliam na garantia de qualidade da solução que está sendo construída.

Monitoramento da Saúde da Aplicação

Monitorar a saúde da aplicação também é fundamental, assim como esse monitoramento pode nos ajudar a ter insights sobre melhorias que podem ser feitas para melhorar ainda mais a qualidade da aplicação.

Abaixo mostraremos um exemplo de tela de monitoramento que é a do App Dynamics, uma poderosa ferramenta de monitoramento e análise de performance de aplicações.

Figura 7 – Monitoramento App Dynamics.



Fonte: https://www.appdynamics.com/blog/product/how-ing-gains-visibility-into-their-complex-distributed-environment/.

1. Aqui visualizamos o status das Business Transactions envolvidas. Estas são transações que têm uma importância maior para os processos de negócio.
2. Aqui visualizamos a saúde do servidor de aplicação.
3. Aqui temos uma visão analíticas das transações dividias entre as que estão normais, lentas, muito lentas, stall (abandonadas, sem resposta) e com erro.
4. Temos um grafo de dependência entre as aplicações databases e serviços externos que estão sendo monitorados.
5. Este gráfico indica o tempo de resposta médio da aplicação ao longo do tempo.
6. Este gráfico indica a quantidade de erros distribuídas ao longo do tempo.
7. E este representa o número de requisições recebidas por minuto ao longo do tempo.

> Capítulo 5

Capítulo 5. Processos Decisórios

Arquitetura por si envolve a tomada de várias decisões para decidir o rumo do desenvolvimento de um software. Estas decisões irão em sua maior parte impactar o software nas suas bases e por isso precisamos de utilizar ferramentas que sejam capazes de nos apoiar a tomá-las. Neste capítulo vamos exemplificar algumas ferramentas que podem ser utilizadas no trabalho do dia a dia do arquiteto para formalizar e apresentar as decisões tomadas durante a construção de um software.

Matriz de Decisão

A matriz de decisão é um mecanismo tabular e bem simples que tem como objetivo auxiliar no processo decisório. A matriz pode ser adaptada a qualquer tipo de decisão que tenhamos que tomar durante as decisões inerentes a um projeto de software que estamos iniciando, seja stack, ferramentas de desenvolvimento, cloud etc.

O processo é bem simples, organizamos nas colunas as opções disponíveis e nas linhas os critérios que queremos analisar. Nas células atribuímos notas para cada critério em relação à opção referente. Esta nota pode ou não ser ponderada, cabendo a cada critério ter o seu peso definido. A partir daí a opção que tiver o maior somatório das notas se tornará a mais indicada a ser adotada.

Abaixo um exemplo de matriz de decisão bem simples (abordaremos uma mais complexa em nosso estudo de caso), em que precisamos escolher qual banco de dados utilizaremos baseados em alguns critérios e notas de 1 a 3, em que 1 é a pior nota e 3 é a melhor nota:

Critérios	Peso	Microsoft Sql Server	MySql	Oracle
Custo	2	2	3	1
Disponibilidade de Profissionais no mercado	1	2	2	1
TOTAL		6	8	3

Podemos ver que nesta situação o MySql seria o mais indicado como banco de dados, considerando os critérios informados.

O mais complexo na matriz de decisão é justificar as notas em cada combinação critério/opção e onde algumas vezes pode ser mais subjetivo. Para mitigar isso podemos ter um anexo um embasamento descritivo para estas notas em questão.

Análise de Premissas e Restrições

Premissas e restrições sempre existirão nos produtos de software que trabalhamos e trabalharemos no futuro. Segundo o PMBOK 5ª Edição (2013) uma premissa é um fator do processo de planejamento considerado verdadeiro, real ou certo, desprovido de prova ou demonstração, enquanto uma restrição é um fator limitador que afeta a execução de um projeto ou processo.

Por sendo uma verdade assumida, a premissa provavelmente irá gerar um risco de o assumido não ser verdade. Como arquitetos algumas premissas serão muitas vezes determinadas por nós, como por exemplo, possuímos uma conta na organização para utilizar uma determinada cloud ou possuímos licença para trabalhar com um determinado sistema de banco de dados.

Ainda mais impactantes que as premissas, certamente trabalharemos com várias restrições nos produtos que iremos desenvolver. Muitas vezes iremos atender a um cliente que possui diversas restrições tecnológicas e cabe ao arquiteto trabalhar para construir utilizando estas restrições. Algumas restrições podem ser, a utilização obrigatória de uma

stack específica, por exemplo, utilizar o datacenter on-premise para publicar ou trabalhar em uma cloud única específica. Algumas restrições tecnológicas impostas podem inviabilizar a qualidade de uma solução, e é importante que isso seja comunicado para os sponsors e responsáveis do software que está sendo desenvolvido.

Estudo de Caso

Matriz de Decisão de Stack

No exemplo a seguir faremos uma análise sobre qual stack escolher para desenvolvimento de uma aplicação mobile considerando alguns critérios. Ao contrário do exemplo anterior documentaremos também o porquê das notas que estamos inserindo.

Iremos desenvolver um App para uma empresa de investimento e será necessário que determinemos qual tecnologia iremos utilizar. É importante ressaltar que o aplicativo deverá ser compatível com Android e IOS desde seu lançamento.

Os critérios que utilizaremos são:

- Suporte a funcionalidades nativas.
- Capacidade de abstração de especialização entre plataformas (ios/android).
- Experiência dos desenvolvedores.
- Custo de execução.

Vamos analisar as seguintes stacks:

- React Native.
- Flutter.

Poderíamos incluir algumas outras formas aqui como, por exemplo, utilizar as stacks nativas de cada app (Swift e Kotlin) ou realizar um desenvolvimento web através de um Progressive Web App (PWA), mas vamos manter apenas nas duas primeiras opções apresentadas. Vamos fazer a análise individual de cada stack, começando por React Native, e seguindo o mesmo esquema de notas de 1 a 3.

React Native		
Critério	Justificativa	Nota
Suporte a funcionalidades nativas	Como no React-Native é bem forte a cultura de open-sourcing, existem componentes com código aberto desenvolvidos pela própria comunidade que permitem o uso de recursos do dispositivo mobile: ▪ Acesso à câmera (frontal e traseira). ▪ Autenticação por biometria. ▪ Leitura de SMS. Além de ferramentas de suporte e análise: ▪ Firebase Analytics. ▪ Criptografia com os padrões ASE e RSA.	3
Capacidade de abstração de especialização entre plataformas	Uma das desvantagens do React-Native é que, devido a forma como o javascript é executado com thread isoladas da thread nativa, pode ser necessário realizar ajustes em alguns códigos nativos. Dessa forma pode ser necessário a atuação de alguns desenvolvedores com conhecimentos de desenvolvimento nativo. Além disso, pode haver casos que a configuração de certas bibliotecas em React-Nativa necessite de especialização de código para as plataformas Android e iOS.	2

Experiência dos desenvolvedores	Ferramentas de depuração de JS no browser: ▪ Chrome Developer Tools e Safari Developer Tools. ▪ Reactotron é uma ferramenta de código aberto utilizada para inspecionar o estado do Redux de aplicações React, sendo possível também a visualização de logs e execução de comandos que restauram o estado de componentes, dentre outras funcionalidades. Hot reload: No React-Native a partir da versão 0.61, temos uma feature chamada "Fast Refresh", disponibilizada pelo React Developer Tools, que trouxe melhorias ao "Hot Reload" de versões anteriores: ▪ Suporta totalmente o React moderno, incluindo componentes funcionais e hooks. ▪ Se recupera normalmente após erros de digitação e outros erros e volta a ser recarregado quando necessário. ▪ Não realiza transformações invasivas de código, portanto é confiável o suficiente para estar ativado por padrão.	3
Custo de Execução	Temos vários desenvolvedores capacitados em Javascript e React, logo eles podem ser utilizados para o desenvolvimento, retirando a necessidade de novas contratações. Os	2

	desenvolvedores precisariam de tempo para aprender o funcionamento do React Native.	

Flutter		
Critério	Justificativa	Nota
Suporte a funcionalidades nativas	Das funcionalidades que utilizam recurso de compomentes do app, temos suporte com o Flutter a: ▪ Uso da câmera. ▪ Autenticação por leitura biométrica. ▪ Leitura de SMS (permitido apenas para Android). Ferramentas de terceios com suporte oficial em flutter (framework): - Firebase Analytics. - Criptografia com os padrões ASE e RSA.	3
Capacidade de abstração de especialização entre plataformas	Os componentes da camada de framework do flutter são por definição abstrações de componentes que serão especializados, no momento da compilação, em componentes nativos de cada uma das plataformas mobile. Componentes como "Widgets", "Rendering", "Animation", "Material", "Cupertino" e "MethodChannel" são exemplos de componentes base do flutter (mantidos pelo time core) nos quais a comunidade de desenvolvimento se baseia e compartilha seus próprios componentes, utilizando para isso o catalogo "Flutter Dev" (também suportada pela própria Google). De forma mais ampla podem haver situações que não existam componentes flutter que resolvam um problema específico que um código	3

	usando uma api nativa resolve. Para isso, o uso do recurso de "Plataform Channel" (baseado nos componentes "MethodChannel" e "BinaryMensage" da camada framework) é a solução, e o código especializado na linguagem nativa é mantido, o que permite ainda a criação de componentes flutter internos que abstraiam essa integração.	
Experiência dos desenvolvedores	Ferramentas para depuração: DevTools permite inspeções layout e estado de UI, diagnóstico de gargalos de performance com interfaces lentas, depuração no nível de código fonte por meio de linha de comando, geração de logs e informações de execução em uma interface gráfica. Hot reload: Permite recarregar o código fonte compilado em uma Dart Virtual Machine em execução, mantendo o estado de todos os componentes da árvore atual de widgets mesmo que na prática os componentes sejam reorganizados na árvore. Alterações no método initState na classe Widget (framework) não funcionaram com o "Hot reload "	3
Custo de Execução	O desenvolvimento em DART não é comum para os desenvolvedores, então haverá um esforço maior de treinamento no início até que o time consiga acelerar para continuar.	1

Agora vamos montar a matriz para analisarmos o resultado:

Critério	Peso	React Native	Flutter
Suporte a funcionalidades nativas	1	3	3
Capacidade de abstração de especialização entre plataformas	1	2	3
Experiência dos desenvolvedores	1	3	3
Custo de Execução	2	2	1
TOTAL		12	11

Baseado na análise, a implementação em React Native foi mais bem pontuada, logo pelo método ela é a solução indicada. Claro que estamos fazendo uma análise fictícia e poderíamos mudar os pesos ou incluir alguma informação que melhore ou piore um indicador, o importante é montar visualmente a matriz para conseguirmos usá-la de guia para a tomada de decisão.

Análise de Custo da Solução

Custo muitas vezes vai ser uma restrição de um projeto. Organizações normalmente definem orçamentos anuais para cada um dos projetos que julgam ser os mais importantes e normalmente esta divisão é uma das principais restrições para o desenvolvimento. Em certos momentos podemos ser requisitados para auxiliar a levantar os custos do projeto. Aqui iremos levantar o custo necessário mensal para provisionar a solução na Azure, usando a calculadora do site.

Fomos contratados para levantar os custos de cloud para implementar uma aplicação que vai conter uma Api Rest, um banco de dados e um front end. O projeto no primeiro momento não irá bancar custos altos pois a ideia é validar o funcionamento e a adoção do produto pelos clientes, com isso precisamos construir uma solução bem enxuta.

Dados estes requisitos temos a seguinte estimativa, que possui 2 app services, um para o frontend e um para o backend, e temos uma database, totalizando um custo de $54,25 mensais para o projeto. Caberia a nós agora a compartilhar esta estimativa com os sponsors do projeto para avaliarem isto em relação ao orçamento.

Figura 8 – Análise de Custo.

Service type	Region	Description	Estimated monthly cost	Estimated upfront cost
App Service	East US	Camada Básico; 1 B1 (1 núcleos, 1.75 GB de RAM, 10 GB de armazenamento) x 730 Horas; Linux SO	$12.41	$0.00
Azure SQL Database	East US	Banco de dados único, DTU modelo de compra, Padrão camada, S1: 20 DTUs, 250 GB de armazenamento incluído por BD, 1 bancos de dados x 730 Horas, 5 GB retenção	$29.43	$0.00
App Service	East US	Camada Básico; 1 B1 (1 núcleos, 1.75 GB de RAM, 10 GB de armazenamento) x 730 Horas; Linux SO	$12.41	$0.00
Support		**Support**	$0.00	$0.00
		Licensing Program	**Microsoft Online Services Agreement**	
		Total	**$54.25**	**$0.00**

> Capítulo 6

Capítulo 6. Riscos e Mitigação de Riscos

Riscos são inerentes a projetos e produtos de software. Sempre teremos situações em que precisaremos avaliar o quão impactante será uma situação de incerteza para o que estamos construindo e precisaremos trabalhar em planos de ação efetivos para tentar reduzir esta incerteza a um grau aceitável. Neste módulo estudaremos sobre riscos e algumas técnicas para lidar com eles em um projeto de software.

O que é um Risco?

Segundo o PMBOK 5ª Edição (2013), risco é um evento ou condição incerta que, se ocorrer, provocará um efeito positivo ou negativo em um ou mais objetivos do projeto tais como escopo, cronograma, custo e qualidade. Um risco pode ter uma ou mais causas, e se ocorrer, pode ter um ou mais impactos. O risco surge da incerteza existente, sejam elas no projeto, na organização ou em qualquer outra parte do projeto.

Os riscos positivos e negativos são frequentemente chamados de oportunidades e ameaças. As oportunidades podem estar relacionadas a uma incerteza em que se ela acontecer terá um impacto positivo no projeto, por exemplo, uma nova oportunidade de negócio que ainda é incerta, mas que pode alavancar a rentabilidade do produto. Já as ameaças são incertezas que se acontecerem terão um impacto negativo no projeto, por exemplo, falta de capacitação técnica em uma determinada tecnologia pode provocar atrasos na entrega do produto e aumentar os custos proeminentes com treinamento e contratação de recursos qualificados.

A forma como as organizações lida com os riscos é algo bem cultural e relacionado com o momento que elas vivem. São diversos os fatores que influenciam o quão risk-taker é uma organização, mas estes fatores podem ser classificados de forma ampla em três tópicos:

- Apetite ao risco: é o grau de incerteza que uma entidade está disposta a aceitar, na expectativa de uma recompensa.

- Tolerância ao risco: é o grau a quantidade ou o volume de risco que uma organização está disposta a tolerar.

- Limite de riscos: se refere às medidas ao longo do nível de incerteza ou nível de impacto no qual uma parte interessada pode ter um interesse específico. A organização aceitará o risco abaixo daquele limite e não tolerará acima dele.

Agir proativamente sobre os riscos, planejar e acompanhar é fundamental para o sucesso de um projeto. Estas atividades devem permear o projeto durante todo o ciclo de vida, pois novos riscos podem surgir. Riscos que foram encontrados no início podem não acontecer ao longo do projeto, assim como riscos não previstos podem acontecer. Avançar um projeto sem focar o gerenciamento dos riscos pode causar mais problemas surgidos em virtude de ameaças não gerenciadas.

Riscos e Desenvolvimento de Software

Devido a sua característica envolvendo muitas incertezas, o desenvolvimento de software é algo que envolve vários riscos para uma organização. Vários fatores permeiam o processo de desenvolvimento de software e com isso vários riscos aparecem ao longo do passar do tempo do desenvolvimento.

Os riscos de um projeto de software podem ser das mais diversas fontes, como por exemplo:

- Capacitação técnica dos desenvolvedores: muitas vezes os desenvolvedores precisam lidar com tecnologias que não estão habituados.

- Requisitos mal definidos: pode levar a várias rodadas de retrabalho, aumentando os custos e o prazo.

- Prazos apertados: muitas vezes o desejo do negócio de lançar um produto para ter uma maior competitividade no mercado se arremete em prazos muito curtos para realização do desenvolvimento.

- Mercado de desenvolvimento aquecido: talvez este seja o maior risco na atualidade, a grande escassez de mão de obra de desenvolvedores no mercado de trabalho está forçando as organizações a ficarem cada vez mais preocupadas em manter seus profissionais de TI, pois todos os dias são bombardeados de mais e mais ofertas de trabalho.

- Constantes mudanças de priorização, isso se arremete também à falta de um objetivo de negócio bem definido para o software e consequentemente o tempo todo as prioridades mudam, assim como o foco dos desenvolvedores, e isso contribui fortemente para improdutividade.

Como arquitetos de softwares, estes riscos sempre vão “bater a nossa porta”, principalmente os riscos técnicos e precisamos procurar formas de mitigar estes riscos para que o desenvolvimento flua com o mínimo distúrbio.

Como identificar Riscos?

Identificar os riscos é o processo de determinação dos riscos que podem afetar o projeto e de suas características (PMBOK, 2013). O principal benefício deste processo é ter documentado os riscos existentes e isto permite que o time consiga atuar preventivamente para que as ameaças sejam evitadas e as oportunidades aconteçam.

Existem várias formas para se identificar os riscos inerentes ao projeto, abaixo vamos listar alguns destes métodos:

- Revisão de documentações: aqui podemos revisar os documentos relacionados ao projeto e através deles identificar os riscos. Estes documentos podem ser atas de reuniões iniciais, especificações de requisitos ou backlog inicial de desenvolvimento.

- Brainstorming: discutir em conjunto com todas as partes interessadas do projeto através de um facilitador, coletar e classificar os riscos que cada um identificar.

- Técnica Delphi: É uma maneira de obter um consenso dos especialistas em riscos em que cada um participa anonimamente. Nela cada um preenche um questionário que solicita ideias dos riscos mais importantes dos projetos e as respostas são redistribuídas entre os demais para comentários. O consenso pode ser obtido após algumas rodadas desse processo.

- Entrevistas: Conversas individuais com partes interessadas, stakeholders ou pessoas especialistas naquele tipo de projeto. Esta hora podemos conversar com outros desenvolvedores e arquitetos que já tenham passado por situações parecidas com o projeto.

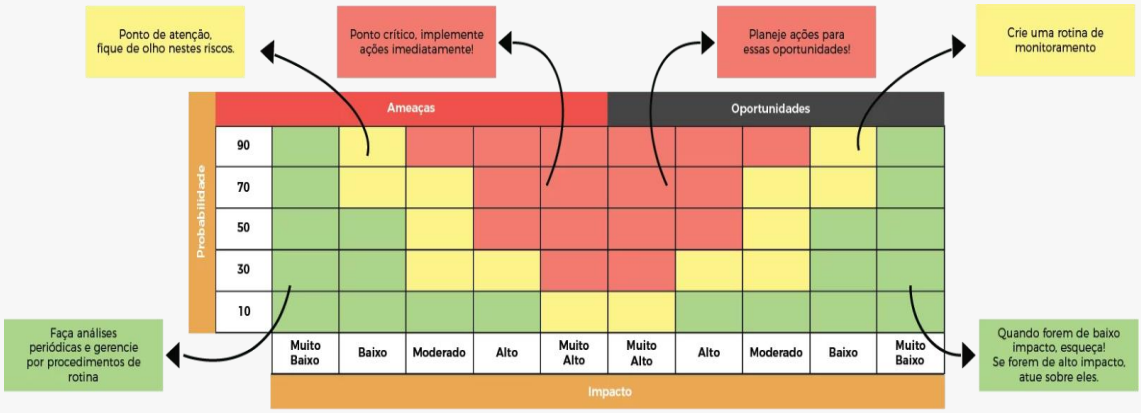
- Análise da causa principal: O objetivo é identificar um problema, descobrir as causas subjacentes e desenvolver ações preventivas para o mesmo.

- Análise SWOT: Esta técnica avalia o projeto na vista de suas forças (Strengths), fraquezas (Weaknesses), oportunidades (Opportunities) e ameaças (Threats). Através dela identificaremos oportunidades baseadas nas forças e ameaças baseadas nas fraquezas e elencar os riscos resultantes.

Como classificar Riscos?

Uma vez que se tenha os riscos identificados, precisamos classificá-los para que levantemos quais iremos atuar primeiro, criando planos de ação. Uma técnica que pode nos ajudar muito nesta situação é a análise de probabilidade e impacto sobre cada um dos riscos através de uma matriz.

Figura 9 – Matriz de Risco



Fonte: https://blogdaqualidade.com.br/o-que-e-uma-matriz-de-riscos/.

Essa matriz de análise de probabilidade e impacto foi modificada para colocarmos seções separadas para oportunidades e ameaças. Assim que distribuirmos nossos riscos nela podemos fazer a seguinte análise.

Ameaças:

- Quanto maior o impacto e maior a probabilidade (Região Vermelha), devemos ter foco na mitigação e em criar ações, o mais rápido possível, para que não aconteçam.
- Riscos na região amarela devem ter atenção para que eles não transitem para a região vermelha e se tornem problemas ainda maiores.
- Ricos na região verde devem ser acompanhados periodicamente dentro das atividades rotineiras do projeto.

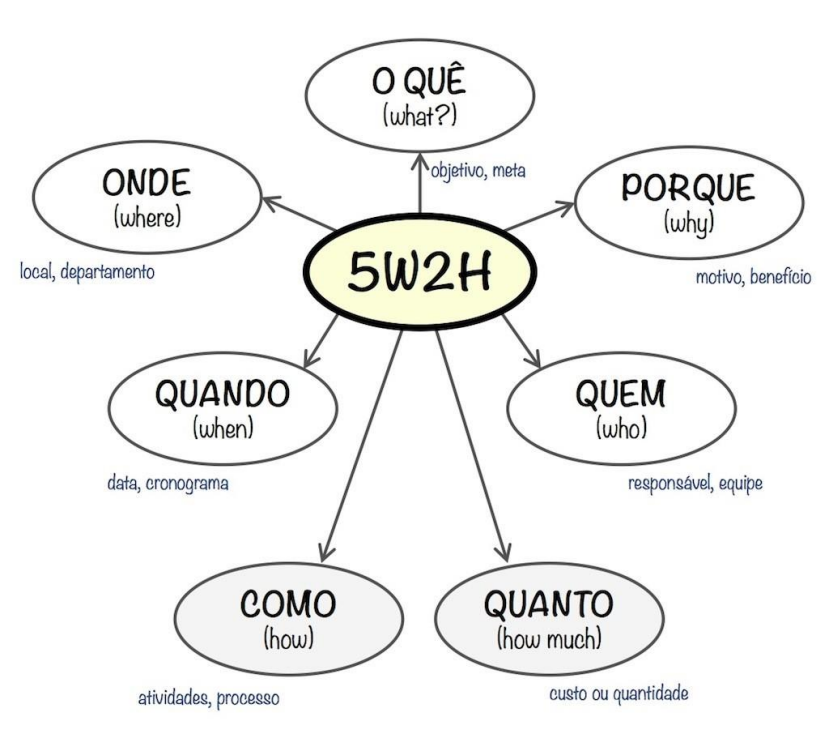
Oportunidades:

- Oportunidades na região vermelha devemos gastar esforços e traçar planos para que aconteçam. Uma boa oportunidade de negócio pode estar aí.
- Oportunidades na região amarela, monitore e tente estudar as de impacto grande para serem reavaliadas na região vermelha.
- Na região verde, se forem de baixo impacto, não perca tempo com elas se for de impacto alto, tente traçar planos de ação para que aconteçam.

Como fazemos planos de ação?

Uma vez os riscos identificados e classificados, precisamos criar planos de ação consistentes para evitar as ameaças e fazer com que as oportunidades aconteçam. Existem várias formas de se fazer o levantamento destes planos de ação, mas aqui vamos focar em uma metodologia bem simples e efetiva que é o 5W2H.

Figura 10 – 5W2H.



Fonte: https://www.treasy.com.br/blog/5w2h/

Esta metodologia constitui em 7 perguntas básicas que precisamos fazer e o conjunto destas respostas serão o plano de ação a ser tomado sobre o risco. Estas perguntas são:

- O quê? (What?): descrição da ação ou atividade que será executada. Isto delimitará o que deve ser planejado fazer.
- Quem? (Who?): quem são as pessoas envolvidas. Estes podem ser stakeholders, desenvolvedores etc. Aqui também será definido os responsáveis pela execução.
- Onde? (Where?): onde a ação será executada. Isto pode ser desde uma localização geográfica até um ponto específico em um código.
- Quando? (When?): aqui responderemos o prazo para execução.
- Como? (How?): descrição da forma de como será feito, etapa a etapa, de uma forma mais detalhada.
- Quanto (How Much?): qual o custo envolvido.
- Por quê? (Why): descrição da razão por tomar esta ação e o quão prioritário isto vai ser em relação a outras.

É importante aqui gastar tempo respondendo estas perguntas para gerar bons planos de ação para os riscos que temos mapeados. Você terá muito mais clareza nas tomadas de decisão em busca de um objetivo. E acredite, vai otimizar tempo e outros recursos.

Estudo de caso

Agora vamos colocar a mão na massa e vamos utilizar este conhecimento que criamos com as seções anteriores em um estudo de caso fictício. Nós deveremos levantar os riscos e fazer a análise 5W2H de alguns destes riscos que levantarmos.

A Eletricity é uma das maiores lojas de venda de produtos e materiais elétricos do Brasil, vendendo materiais tanto para o cliente final quanto no atacado para outras lojas. Há um grande desejo da diretoria de revitalizar o site para ficar mais moderno e mais atraente para os clientes pessoa física. Hoje o site utiliza uma plataforma proprietária que já não recebe atualizações a mais de 3 anos e foi descontinuada pelo fornecedor. Tudo está instalado na infraestrutura local da empresa, que possui um data center interno. O banco de dados também está armazenado localmente nos servidores do datacenter. O pessoal de negócio também colocou as seguintes premissas:

- Existe uma integração direta com o ERP da loja e do estoque que automaticamente cadastra os produtos. Porém utiliza um protocolo de comunicação bem antigo chamado EDI via socket. Existe uma documentação antiga que explica como foi feita essa integração.

- Existe um desejo de que este projeto seja o piloto para utilização de infraestrutura em cloud. Não existem outros produtos de software em cloud hoje na Eletricity.

- Não existem profissionais de TI qualificados na empresa hoje para fazer o desenvolvimento, e a diretoria quer que sejam contratados profissionais capacitados que começaram a compor uma nova área de desenvolvimento de software na Eletricity.

Você foi o primeiro contratado para atuar com o papel de Arquiteto de Soluções na Eletricity, já tem uma bagagem longa de trabalho com cloud computing e a diretoria chegou com este desejo para você, fazer uma análise dos riscos técnicos que podem envolver o projeto.

Analisando este requisito acima, podemos identificar no mínimo uma tonelada de riscos se formos muito minuciosos, com isso, vamos nos concentrar apenas nos abaixo por enquanto:

1. O formato EDI é bem antigo, será difícil achar profissionais com essa experiência.

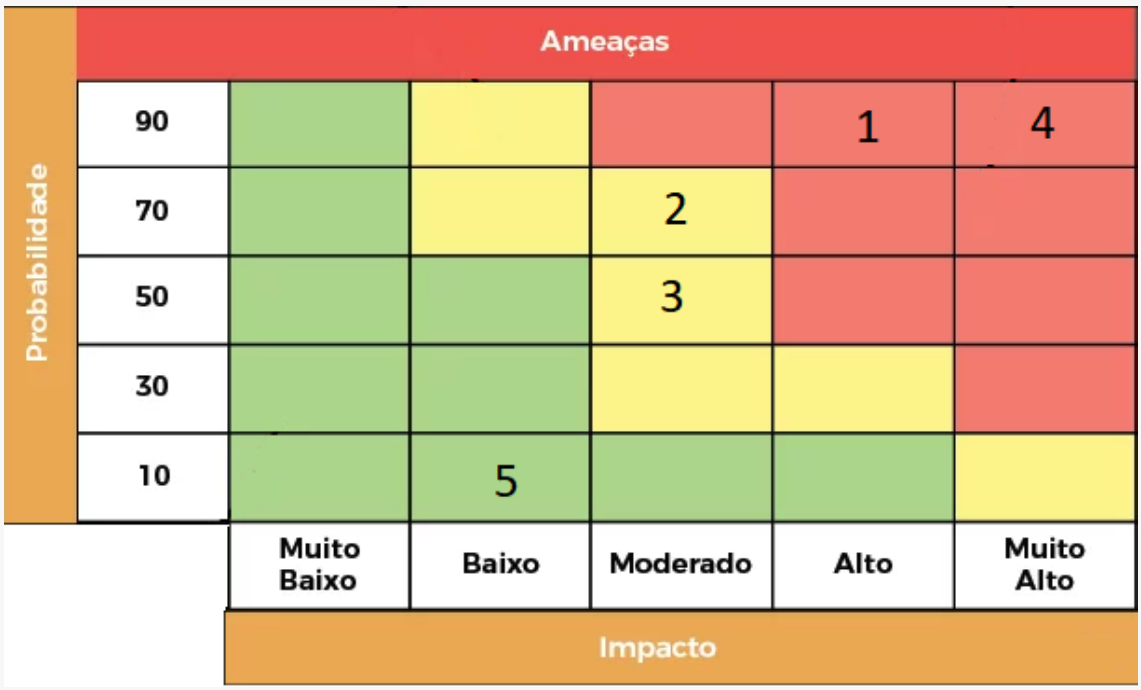
2. Não existe uma cultura nem infraestrutura pronta para trabalhar com qualquer cloud. Pode haver resistência do pessoal que trabalha com o datacenter.

3. Devemos contratar profissionais capacitados e motivados para trabalhar nessa transformação. Com isso o início será muito conturbado e o mercado está muito aquecido.

4. O novo site precisa continuar comunicando com a base de dados on-premise. Por isso precisamos ter uma comunicação segura entre o datacenter e a cloud.

5. Podemos ter indisponibilidade dos consultores das clouds públicas para fazer uma proposta de preço.

Figura 11 – Matriz de Risco



Fonte: https://blogdaqualidade.com.br/o-que-e-uma-matriz-de-riscos/.

Pronto, classificamos nossos riscos:

- A probabilidade é alta de não achar um desenvolvedor que saiba trabalhar com padrão EDIFACT e pode trazer um impacto grande para o desenvolvimento, mas podemos estudar bastante a documentação do ERP para entender como funciona, por isso o impacto não é “Muito Alto” e é “Alto”.

- É normal que o pessoal de infraestrutura tenha resistência e receio em começar a fazer implantações em cloud computing, mas temos o suporte da diretoria neste assunto.

- Está escasso no mercado o profissional desenvolvedor, mas ainda não está impossível de encontrar estes funcionários. O risco que temos é de gastar mais tempo procurando mão de obra antes de iniciar o desenvolvimento. Podemos por exemplo contratar uma empresa de consultoria inicialmente e depois irmos contratando desenvolvedores.

- Temos uma comunicação cloud -> on-premise que precisa ser tratada com cautela e precisa funcionar para o sucesso do projeto.

- A indisponibilidade de consultores das clouds públicas pode acontecer, mas como existe uma competitividade gigante entre elas, os consultores ficam correndo atrás de novos clientes com frequência.

Pronto, agora temos nossos riscos mapeados e classificados na matriz. Para demonstrar vamos escolher os piores riscos e traçar um plano de ação para cada um usando o método 5W2H.

Risco 1

O quê?	Conhecimento deficiente no protocolo EDI
Quem?	Arquiteto de Soluções e RH
Onde?	Eletricity, no projeto de revitalização do site
Quando?	Nos próximos dois meses
Como?	1. O RH irá procurar por pessoas com conhecimento em EDI e contratar uma consultoria de recrutamento e seleção para acelerar o processo. 2. O Arquiteto de Soluções irá estudar a documentação da integração dos ERP's para depois repassar aos novos contratados. Assim como fazer provas de conceito para validar o funcionamento.
Quanto?	Teremos um custo adicional de R$3000 para contratar uma consultoria de recrutamento e seleção para nos auxiliarem a buscar os profissionais.
Por quê?	Isto é um requisito crítico para o sistema e precisamos priorizá-lo na fase de concepção para não levar essa pendência quando o projeto já estiver em andamento.

Risco 4

O quê?	Criar uma comunicação segura entre Cloud e On-Premise.
Quem?	Arquiteto de Soluções e Time de infraestrutura.
Onde?	Datacenter da Eletricity.
Quando?	Nos próximos dois meses.
Como?	• O arquiteto de soluções irá estudar uma arquitetura de referência para este tipo de

	comunicação, serão feitas provas de conceito para validar o funcionamento. • Faremos essa passagem de conhecimento para o time de infraestrutura executar a alteração. • Contrataremos uma consultoria de segurança da informação para validar a nossa solução implementada.
Quanto?	Teremos um custo com a consultoria de segurança, ainda a ser confirmado.
Por quê?	Com a entrada da Lei Geral de Proteção de Dados, a segurança dos dados é extremamente importante para que a Eletricity não sofra com vazamentos de informação.

> Capítulo 7

Capítulo 7. Frameworks Arquiteturais

Frameworks arquiteturais são conjuntos de ferramentas com o objetivo de auxiliar as organizações a formatarem a arquitetura corporativa de modo que o negócio e a tecnologia consigam trabalhar de uma maneira unificada através de processos bem definidos. Estes frameworks vão ajudar a definir estes processos dentro da organização.

Neste capítulo abordaremos quatro frameworks arquiteturais, sendo eles:

- Zachman Framework, que pode ser considerado o precursor da Arquitetura Corporativa.
- TOGAF, que é o mais amplamente utilizado pelas organizações e mais importante que isso, o que é mais evoluído e atualizado ao longo dos anos.
- Também falaremos de forma mais sucinta do DoDAF e FEAF que foram amplamente utilizados pelo governo dos Estados Unido.

Zachman

John Zachman é considerado o “pai” da Arquitetura Corporativa. Foi ele quem delineou a disciplina, inventou o termo “Enterprise Architecture” e criou o primeiro framework, que leva seu nome.

O Modelo de Zachman evoluiu pela observação de como o trabalho era planejado e realizado em disciplinas que existiram durante séculos. É baseado na filosofia de que os mesmos conceitos fundamentais existem dentro do ambiente de informação e que os aplicando pode disponibilizar sistemas e outros produtos, com o mesmo poder de duração e de confiança de edifícios e máquinas de qualidade.

Este modelo reconhece que os sistemas de informação têm de se relacionar com o negócio. No negócio as pessoas têm diferentes perspectivas ou papéis e por necessidades diferentes. As necessidades em cada perspectiva podem ser expressas pelo entendimento de cada uma de uma série de dimensões ou abstrações. Um entendimento mais profundo destas necessidades ajuda a construir um sistema de informação que pode ir ao encontro dessas necessidades.

Em sua forma atual, o framework de Zachman consiste em uma matriz de 6 colunas por 6 linhas. As colunas correspondem às clássicas perguntas 5W1H (What/Who/Where/When/Why/How) aplicadas à organização. As colunas, portanto, referem-se aos diferentes aspectos sobre a organização que precisam ser conhecidos:

- What: sobre o que a organização precisa de informação? De que ela trata? Normalmente, essa coluna representa dados mantidos pela organização.
- How: Como a organização funciona? Como ela processa seus dados? Esta coluna normalmente refere-se a processos e funções da organização.
- Where: Onde as coisas acontecem? Aqui vão informações geográficas, de localização etc.
- Who: Quem está na organização e quem faz o quê? Informações sobre pessoas e estruturas organizacionais estão aqui.
- When: Quando as coisas acontecem? Questões relativas ao tempo aparecem aqui.
- Why: Por que as coisas acontecem? Aqui vão as informações relativas às motivações da organização, incluindo seus planos estratégicos de negócio.

As linhas da matriz referem-se aos diferentes pontos de vista e níveis de detalhe relativos à informação que descreve a organização:

- A primeira linha contém o escopo e o contexto, e representa o ponto de vista do estrategista como teorizador sobre a organização. Normalmente, contém informação relevante para o planejamento estratégico de alto nível e, é claro, o próprio conteúdo da Estratégia da organização.

- A segunda linha contém os conceitos de negócio, representando a visão da liderança executiva (vistos como proprietários dos processos de negócio e informações relacionadas). Contém tipicamente descrição detalhada da organização no nível de processos de negócio.

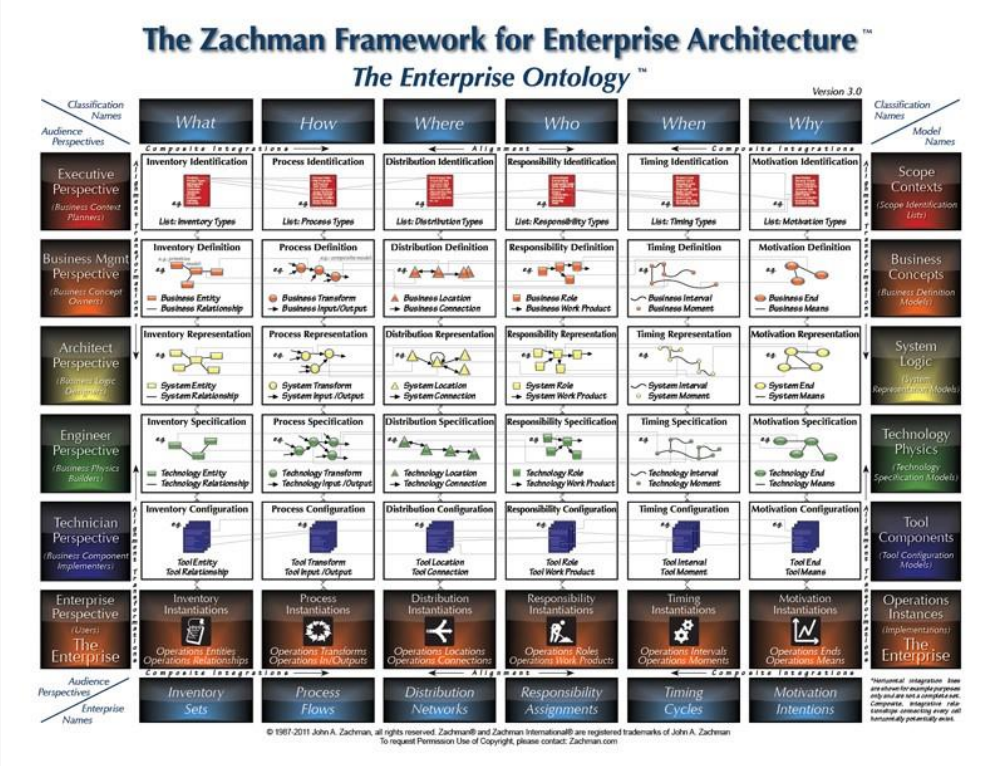
- A terceira linha contém informações sobre os sistemas de informação (nível lógico), com a visão dos arquitetos de sistemas (designers).

- A quarta linha contém informações sobre a infraestrutura tecnológica (nível físico) da organização, sendo o ponto de vista dos engenheiros enquanto construtores.

- A quinta linha refere-se à descrição dos componentes que a organização utiliza para operar, sendo a visão dos técnicos-implementadores.

- A sexta e última linha representa as operações propriamente ditas da organização, instanciadas pelos seus colaboradores participantes.

Figura 12 – Matriz Zachman Framework.



Fonte: https://www.zachman.com/about-the-zachman-framework.

TOGAF – The Open Group Architecture Framework

O Open Group Architecture Framework (TOGAF) é um framework que ajuda a construir uma arquitetura de TI corporativa que oferece uma estrutura de alto nível para o desenvolvimento de software. O TOGAF ajuda a organizar o processo de desenvolvimento por meio de uma abordagem sistemática que visa reduzir erros, manter cronogramas e orçamento, além de alinhar a TI com as unidades de negócios para produzir resultados de qualidade. A característica distintiva do TOGAF (além do fato de ser aberto) é a disponibilização de uma metodologia amplamente customizável para orientar os esforços de arquitetura.

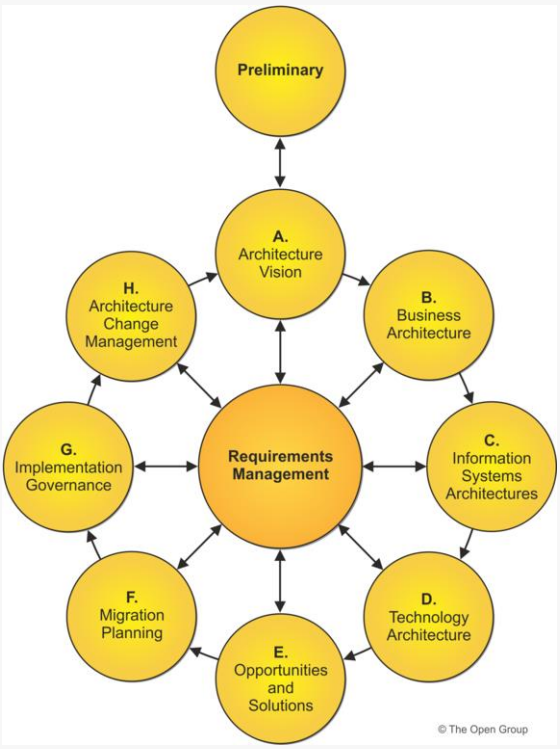
O TOGAF divide a arquitetura corporativa em quatro pilares básicos:

- Business Architecture (Arquitetura de Negócio): define a estratégia de negócio, governança, organização e os processos chaves do negócio.

- Data Archtecture (Arquitetura de Dados): descreve a estrutura física e lógica dos ativos de dados da organização, bem como os recursos a serem gerenciados.
- Application Architecture (Arquitetura de Aplicação): provisiona um blueprint para que as aplicações individuais sejam publicadas, as interações e o relacionamento delas com os principais processos de negócio da organização.
- Technology Archiitecture (Arquitetura Técnica): descreve todo o hardware, software e infraestrutura de TI necessários para desenvolver e implantar aplicativos de negócios.

O coração do TOGAF é o Architecture Development Method (ADM), que estabelece uma metodologia para desenvolvimento e manutenção de uma Arquitetura Corporativa. A representação gráfica do mesmo pode ser lida no seguinte diagrama:

Figura 13 – TOGAF ADM.



Fonte: https://www.opengroup.org/togaf.

Cada círculo no diagrama representa uma fase no ciclo do ADM e cada uma destas fases é dividida paços bem definidos. Falaremos adiante sobre cada uma destas fases de forma sucinta e o que elas abordam:

- Fase Preliminar: é aquela na qual “colocamos em pé” o esforço de arquitetura, estabelecendo a equipe de arquitetura e definindo o método e metamodelo customizados a serem usados em nosso esforço de arquitetura. É aqui também que escolhemos ferramentas de repositório, definimos os processos de Governança da Arquitetura e obtemos o Patrocínio necessário para o esforço de arquitetura.

- Fase A – Visão Arquitetural: Trata-se de estabelecer uma visão de como deve ser nossa arquitetura futura para atender às metas estratégicas de negócio, que são a principal entrada para esta fase. O resultado desta fase é um Documento de Visão da Arquitetura, que documenta onde a organização quer chegar com sua arquitetura para viabilizar o cumprimento das metas estratégicas, e um Plano de Projeto para a execução desta “rodada” do ADM

- Fase B – Arquitetura de Negócio: Desenvolver e documentar a Arquitetura de Negócio desejada (alvo), que irá descrever o que a organização necessita para operacionalizar os objetivos de negócio e responder aos direcionamentos definidos na Visão Arquitetural. O resultado da fase é o detalhamento das necessidades em termos de processos de negócio para atender às metas estratégicas, bem como um gap analysis que nos diz qual é a distância entre nossa situação atual (AS-IS) e a arquitetura em que queremos chegar (TO-BE).

- Fase C – Arquiteturas de Sistemas de Informação: Nesta fase, identificamos os sistemas e dados necessários para atender à situação futura de processos de negócio desenhada na fase anterior,

bem como nossa situação atual e a distância a ser percorrida (gap analysis).

- Fase D – Arquitetura de Tecnologia: Esta fase se ocupa de documentar as necessidades futuras em termos de infraestrutura tecnológica para atender às necessidades de sistemas e dados identificados na fase anterior. Mais uma vez, identificamos também nossa situação atual e a distância a ser percorrida.

- Fase E – Oportunidades e Soluções: Nesta fase será gerado a versão inicial completa do Roadmap de Arquitetura, baseado nos gaps identificados nas fases B, C e D. Identificaremos também projetos e atividades que deverão ser realizados para atingir os objetivos definidos no Roadmap. O resultado é um portfólio de projetos para atingir a arquitetura desejada.

- Fase F – Migration Planning: Aborda o planejamento detalhado da migração, ou seja, como passar do estado atual para a Arquitetura Alvo, certificando que o valor do negócio e o custo dos pacotes de trabalho e arquiteturas de transição são entendidos pelos stakeholders.

- Fase G – Governança de Implementação: Nesta etapa o principal é a realização de revisões de conformidade com o objetivo de garantir que as atividades definidas no portfólio estão sendo executadas de acordo com a arquitetura proposta.

- Fase H – Gestão de Mudanças na Arquitetura: Trata-se de acompanhar no dia a dia a continuidade da relevância da arquitetura implantada na Fase G às necessidades estratégicas da organização. Mudanças no Ambiente de Negócios e na Estratégia exigirão mudanças na arquitetura, e o processo usado nesta fase deve ser capaz de separar pequenas de grandes mudanças. As grandes

mudanças, tipicamente, exigirão a reentrada no ciclo do ADM, ou seja, o estabelecimento de um novo projeto, a ser iniciado novamente na Fase A.

- Gestão de Requisitos: Esta atividade encontra-se – literalmente – no "centro" do ADM, significando que cada uma das demais fases do ADM ao mesmo tempo gera novos requisitos de arquitetura e utiliza como entrada os requisitos de arquitetura previamente identificados.

É importante frisar que as fases e etapas colocadas pelo TOGAF não são necessariamente uma "receita de bolo" a qual um time de arquitetura corporativa deverá seguir "by the book". Ele deve ser analisado como um conjunto de princípios que irá auxiliar à corporação definir um processo para criação e evolução dessa arquitetura corporativa.

DoDAF – Department of Defense Architecture Framework

O DoDAF é uma estrutura de arquitetura desenvolvida para o Departamento de Defesa do Estados Unidos (DoD) que fornece uma infraestrutura de visualização para preocupações específicas dos stakeholders através de vários pontos de vista organizados por várias visões. Estas visões são artefatos para visualização e entendimento do amplo escopo e complexidades de uma descrição de arquitetura por meios tabulares, estruturais, comportamentais, ontológicos, temporais, gráficos probabilísticos, ou meios conceituais.

O DoDAF fornece uma estrutura básica para desenvolver e representar descrições de arquitetura que garantem um denominador comum para a compreensão, comparação e integração de arquiteturas em fronteiras organizacionais, conjuntas e multinacionais. Ele estabelece definições de elementos de dados, regras e relacionamentos e um conjunto de linha de base de produtos para o desenvolvimento consistente de sistemas integrados ou arquiteturas federadas. Essas descrições de

arquitetura podem incluir famílias de sistemas (FoS), sistemas de sistemas (SoS) e recursos centrados na rede para interoperar e interagir no ambiente de não combate.

O objetivo do DoDAF é definir conceitos e modelos utilizáveis nos seis processos principais do DoD:

- Integração e Desenvolvimento de Capacidades Conjuntas (JCIDS).
- Planejamento, Programação, Orçamento e Execução (PPBE).
- Sistema de Aquisição de Defesa (DAS).
- Engenharia de Sistemas (SE).
- Planejamento Operacional (OPLAN).
- Gerenciamento do Portfólio de Capacidades (CPM).

Além disso, os objetivos específicos do DoDAF 2.0 eram:

- Estabelecer orientação para o conteúdo da arquitetura em função do propósito - “adequado para o propósito”.
- Aumentar a utilidade e a eficácia das arquiteturas por meio de um modelo de dados rigoroso - o Metamodelo DoDAF (DM2) - para que as arquiteturas possam ser integradas, analisadas e avaliadas com mais precisão.

Mais informações acerca do DoDAF podem ser encontradas no site oficial (https://dodcio.defense.gov/library/dod-architecture-framework/).

FEAF – Federal Enterprise Architecture Framework

Foi lançado em 1999 com intuito de promover o desenvolvimento compartilhado para processos federais comuns, interoperabilidade e compartilhamento de informações entre agências federais e outras

entidades governamentais nos Estados Unidos. Em 2013 foi lançada uma segunda versão para descrever ferramentas que auxiliassem o governo a implementar a Common Approach to Federal Enterprise Architecture.

O FEAF em seis modelos de referência interconectados através do Consolidated Reference Model (CRM), cada um relacionado a um domínio sub-arquitetural do framework. São eles:

- Strategy Domain – Performance Reference Model (PRM).
- Business Domain – Business reference Model (BRM).
- Data Domain – Data Reference Model (DRM).
- Applications Domain – Application Reference Model (ARM).
- Infrastructure domain – Infrastrucuture Reference Model (IRM).
- Security Domain – Security Reference Model (SRM).

> Capítulo 8

Capítulo 8. ArchiMate X UML X BPMN

Boa parte do nosso tempo trabalhando como arquitetos será fazendo desenhos que representarão o nosso software de uma forma mais visual. Existem várias notações que são utilizadas para isso e várias ferramentas. A seguir citaremos alguns exemplos de notações que são usadas frequentemente.

ArchiMate

O Archimate é uma linguagem visual de modelagem que utiliza elementos visuais para apoiar a descrição, a análise e a comunicação da Arquitetura Corporativa e que foi desenvolvida e mantida pelo *"The Open Group".* O Archimate tem um *fit* natural para se trabalhar juntamente ao TOGAF uma vez que são sustentados pelo mesmo grupo.

O Archimate é uma linguagem ampla e abrangente que foi projetada para auxiliar o arquiteto corporativo a descrever a arquitetura corporativa, de forma completa e com pouca ambiguidade, em seus principais domínios.

Os conceitos principais e elementos do ArchiMate são representados em seu *Core Framework.* Ele consiste em três camadas e três aspectos. Isso cria uma matriz de combinações, sendo que cada camada possui sua estrutura passiva de comportamento e estrutura ativa.

As camadas são:

- Business Layer: se trata de processos de negócio, serviços, funções e eventos de negócio. Esta camada oferece produtos e serviços para clientes externos que entram em contato com a organização através de processos de negócio.
- Application Layer: são as aplicações de software que suportam os componentes no negócio com serviços de aplicação.

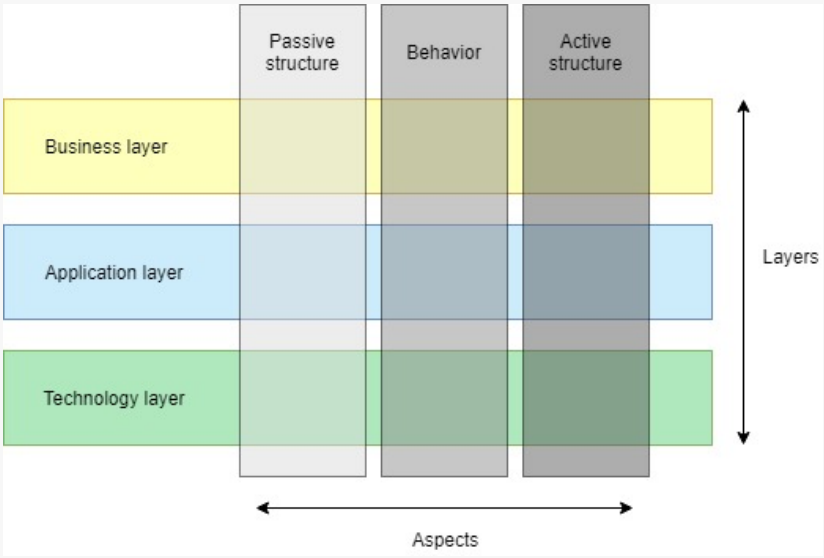
- Tecnology Layer: lida com o hardware e infraestrutura de comunicação para suportar a camada de aplicação. Esta camada oferece todo recurso computacional necessário para executar a aplicação.

Os aspectos são:

- *Passive Structures*: é o conjunto de entidades onde as ações são conduzidas. Na *Business Layer* podem ser objetos de informação, na *Application Layer* podem ser os dados e na *Tecnology Layer* podem incluir objetos físicos.
- *Behavior*: se refere aos processos e funções executadas pelos atores. Elementos estruturais são designados para elementos comportamentais para mostrar quem ou o que mostra o comportamento.
- *Active Structure*: é o conjunto de entidades que mostra algum comportamento, como atores de negócio, dispositivos ou componentes de aplicação.

Com isso chegamos à seguinte matriz:

Figura 14 – ArchiMate Core Framework.



Fonte: https://en.wikipedia.org/wiki/ArchiMate

Para aprender a usar o Archimate, uma ferramenta compatível é necessária, sendo que uma das mais populares é o Archi. Para uma pegada mais prática é o case fictício disponibilizado pelo The Open Group, Archi Surance.

UML

A linguagem UML é na maioria dos casos, a primeira linguagem de modelagem de software que aprendemos quando estamos estudando processo de desenvolvimento de software na universidade.

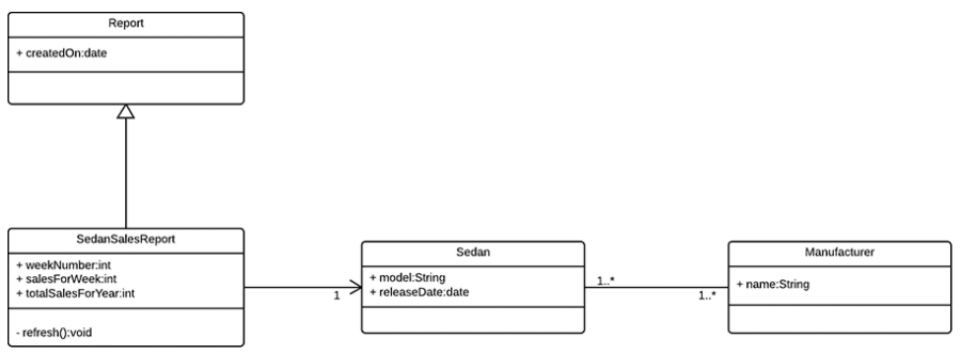
A Linguagem Unificada de Modelagem (do inglês, *Unified Modeling Language)* é uma linguagem para a elaboração da estrutura de projetos de software e que pode ser utilizada para visualização, especificação, construção e documentação de artefatos que façam uso de sistemas complexos de software. Muitos desenvolvedores atualmente podem ter um olhar um tanto desconfiado sobre a linguagem UML, pois muitas vezes se arremete a formas antigas e exaustivas de documentação de software, mas ela ainda é uma linguagem bem eficiente na representação de estruturas complexas de software.

Existem diversos diagramas que são definidos pela linguagem UML e estes diagramas são divididos em dois grandes grupos:

- Diagramas estruturais: são diagramas utilizados para modelar aspectos estáticos e de estrutura do software. Nestes diagramas representaremos as estruturas de classes do software, interface com outros softwares e componentes pouco vulneráveis a mudanças. Os diagramas estruturais são:
  - Diagrama de classes.
  - Diagrama de objetos.
  - Diagrama de componentes.

- Diagrama de implantação.
- Diagrama de pacotes.
- Diagrama de estrutura

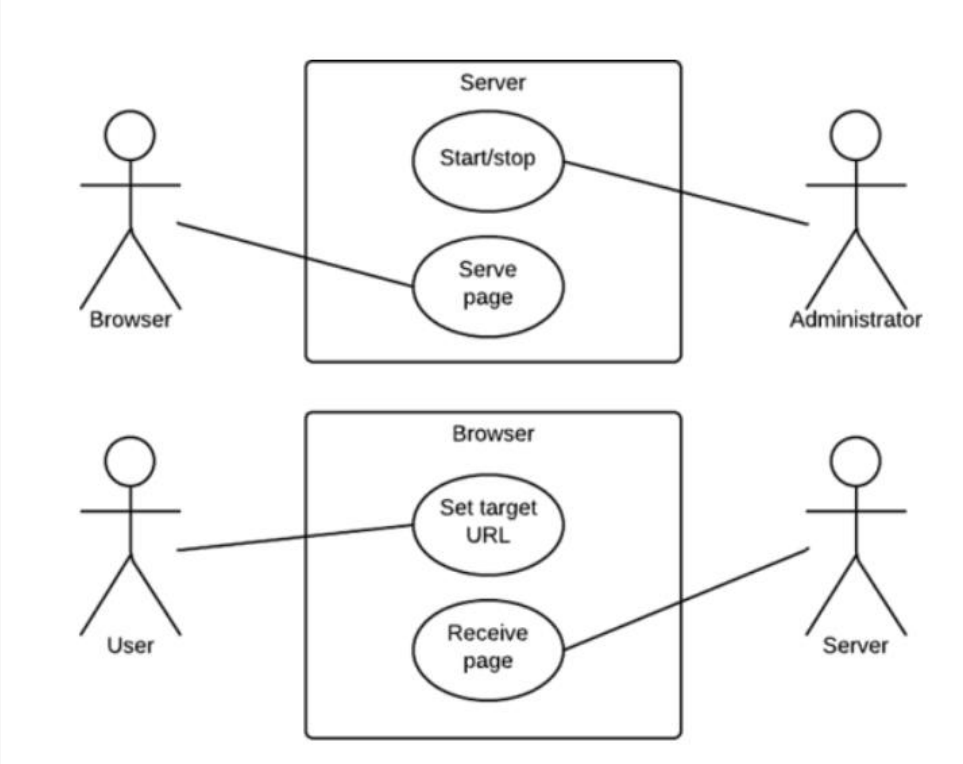
Figura 15 – Diagrama de Classes.



Fonte: https://www.lucidchart.com/pages/pt/o-que-e-uml.

- Diagramas comportamentais: são diagramas utilizados, ao contrário do anterior, para modelar aspectos dinâmicos e de interação que envolvem o software. Aqui representaremos ações de usuário, dinâmicas de troca de mensagens dentre outras. Os diagramas comportamentais são:
  - Diagrama de casos de uso
  - Diagrama de sequência
  - Diagrama de colaboração
  - Diagrama de estados
  - Diagrama de atividade

Figura 16 – Diagrama de caso de uso.



Fonte: https://www.lucidchart.com/pages/pt/o-que-e-uml

Diagramas UML serão extremamente úteis quando precisarmos representar a arquitetura de um sistema em seus mais diversos níveis de entendimento e também servirão para documentar as nossas decisões sobre uma arquitetura.

BPMN

BPMN é mais um tipo de notação que iremos encontrar muitas vezes durante as atividades de arquitetura. Não é necessariamente utilizada para documentar e apresentar desenhos arquiteturais, mas servirá muitas vezes como uma fonte de informação importante para a arquitetura que estejamos desenvolvendo para atender a um determinado software que irá atender a um determinado requisito de negócio.

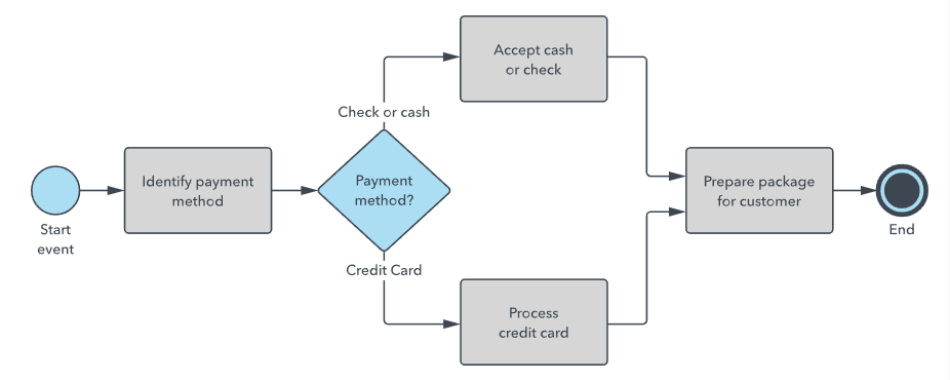
A *Business Process Model and Notation,* é um método através de fluxogramas de documentar de ponta a ponta um processo de negócio. Falando em alto nível, a BPMN auxilia através de uma representação visual

e prática das etapas um processo de negócio para os participantes e outras partes interessadas. Ela fornece uma linguagem padrão comum para todas as partes interessadas, seja técnica ou não técnica: analistas de negócios, participantes do processo, gerentes e desenvolvedores.

Os elementos de uma BPMN possuem quatro categorias básicas:

- Objetos de Fluxo: estes são eventos, atividades e gateways. Eles são os principais elementos descritivos dentro da BPMN.
- Objetos de conexão: conectam os objetos de fluxo uns aos outros e a outras informações. Podem ser fluxos de sequência, fluxos de mensagem ou associações.
- Piscina e raia: Representa os participantes em um processo. Uma determinada piscina pode estar em uma empresa ou em área diferente, mas ainda envolvida no processo.
- Artefatos: São informações adicionais que são adicionadas para acrescentar detalhes ao diagrama para facilitar o entendimento das partes interessadas. Eles podem ser objetos de dados, um grupo (que pode ser usado para agrupar atividades) e anotações.

Figura 17 – Exemplo de diagrama BPMN



Fonte: https://www.lucidchart.com/pages/pt/o-que-e-bpmn.

> Capítulo 9

Capítulo 9. Ciclo de Vida de Desenvolvimento

O ciclo de vida de um software compreende os processos e atividades que envolvem o software desde a sua concepção passando pela operação e manutenção até o ponto deste software cair em desuso e ser substituído por um novo. A critério de comparação é o mesmo cenário que o ciclo da vida na natureza.

Existem vários modelos de ciclo de vida de desenvolvimento e um grande esforço para garantir que esta engrenagem gire o tempo inteiro. Nas sessões a seguir falaremos de alguns modelos de ciclo de vida e do esforço para gerenciá-los.

Modelos do Ciclo de Vida de Desenvolvimento

Os modelos de ciclo de vida são padrões e referências que servem como guia para o processo de desenvolvimento. Os modelos especificarão etapas, atividades-chave e a sequência que estas devem acontecer, assim como os artefatos que devem ser gerados nestas etapas.

Em seguida falaremos de alguns modelos de ciclo de vida:

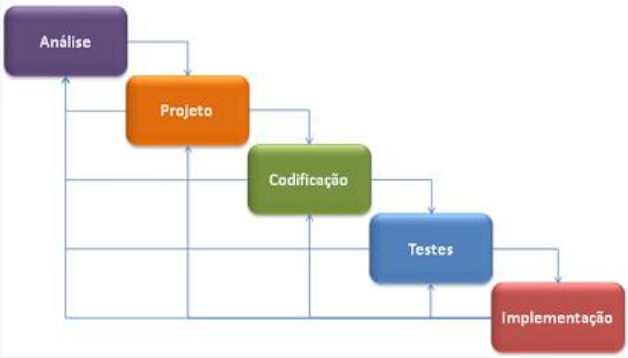
- Cascata.
- Espiral.
- Incremental.

Modelo Cascata

É o modelo de ciclo de vida mais clássico do desenvolvimento de software, em que todo o processo de desenvolvimento é dividido em etapas bem definidas e sequenciais, sendo que uma etapa só se inicia após o término da anterior.

O grande problema aqui é que muito provavelmente veremos uma versão do software só nas etapas finais do desenvolvimento o que deixa a flexibilidade a mudanças durante o processo serem muito custosas, necessitando muitas vezes retornar etapas anteriores para dar continuidade.

Figura 18 – Modelo Cascata.

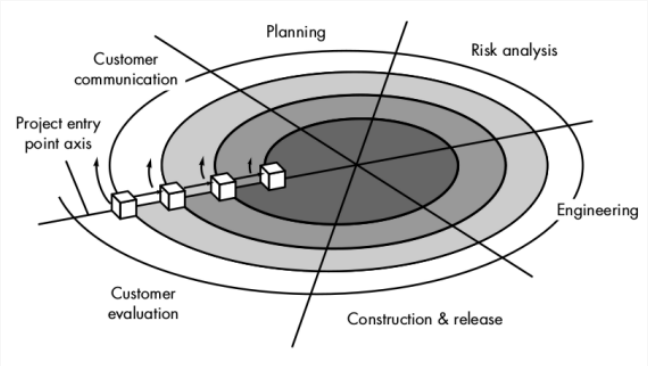


Fonte: http://www.macoratti.net/17/09/net_slcd1.htm

Modelo Espiral

No modelo espiral as fases são tratadas como ciclos, sendo que ao término de cada um teremos uma versão pronta do software operável. O cliente usa o software no seu ambiente operacional, e como feedback, esclarece o que não foi bem entendido e dá mais informações sobre o que precisa e sobre o que deseja (ou seja, mais requisitos). A dificuldade deste modelo é lidar com falhas e bugs que podem acontecer durante a operação.

Figura 19 – Modelo Espiral.



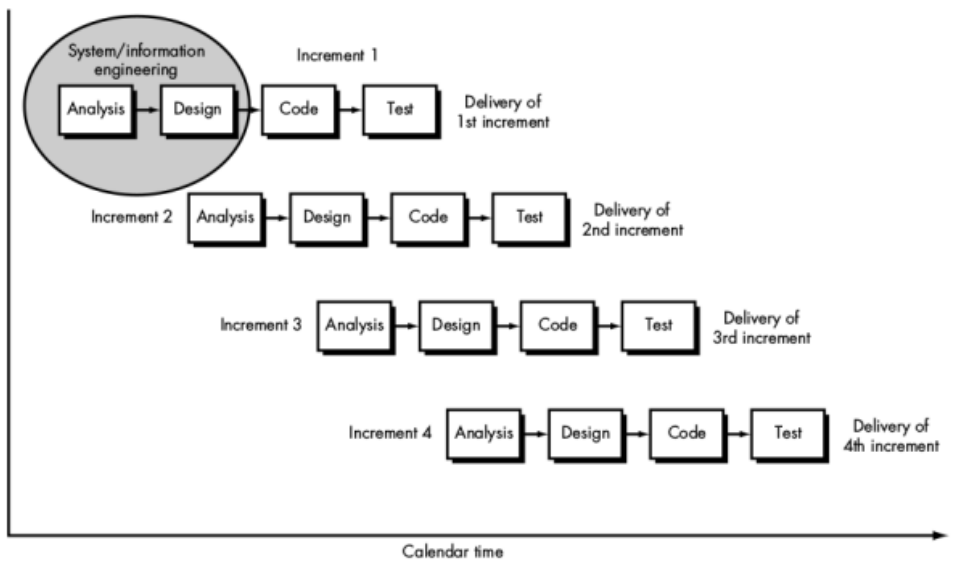
Fonte: Engenharia de Software (PRESSMAN, 2001).

Modelo Incremental

Combina elementos do modelo Cascata Linear com uma ideia iterativa em que incrementos ao software são desenvolvidos de forma paralela e são integrados ao software ao final do desenvolvimento.

Para a abordagem incremental funcionar é necessário que exista um mecanismo para dividir os requisitos do software em partes que serão alocadas em cada ciclo de desenvolvimento de acordo com o grau de importância de cada um.

Figura 20 – Modelo Incremental.



Fonte: Engenharia de Software (PRESSMAN, 2001).

Gerenciamento do Ciclo de Vida (ALM)

O gerenciamento do ciclo de vida de aplicações (*Aplication Lifecycle Management - ALM*) abrange as pessoas, as ferramentas e os processos que gerenciam o ciclo de vida de uma aplicação, desde o conceito até o fim do ciclo.

O ALM se apoia em três grandes pilares:

- Pessoas: o time que está envolvido no ciclo de vida da aplicação atuando desde o nível estratégico até as pessoas que estão trabalhando no desenvolvimento diretamente com o código.

- Processos: boas práticas, padrões e orientações que orientam o desenvolvimento e manutenção do software.

- Ferramentas: todo arcabouço técnico envolvido na construção do software, sejam eles linguagem, servidores, ferramentas de deploy etc.

Várias disciplinas compõem o ALM que em algum momento já foram separadas em processos de desenvolvimentos legados como o Cascata. Estas disciplinas são gerenciamento de projetos, gerenciamento de requisitos, desenvolvimento, controle de qualidade, implantação e operação.

Estas disciplinas estando integradas através do ALM conseguimos promover a entrega contínua de aplicações e atualizações frequentes aplicando abordagens ágeis e de DevOps.

> Capítulo 10

Capítulo 10. Metodologias ágeis x DevOps x Arquitetura de Software

Scrum

O Scrum é um framework que ajuda pessoas, equipes e organizações a gerar valor através de soluções adaptativas para problemas complexos. Incialmente o Scrum estava associado apenas a processos de desenvolvimento de software, mas com a evolução e a aplicação de métodos ágeis nas demais áreas de conhecimento, ele se tornou uma poderosa ferramenta para suporte no desenvolvimento de qualquer produto e também uma forma de gerenciamento de projetos complexos.

Existem valores que permeiam o Scrum e que são essenciais para uma boa condução do projeto/produto que o adota:

- Coragem: os membros da equipe devem ter coragem para fazer o certo e trabalhar através dos problemas.
- Foco: todos estão focados no trabalho e objetivos da sprint atual.
- Comprometimento: as pessoas estão pessoalmente comprometidas com os objetivos da equipe.
- Respeito: toda a equipe é respeitada para ser capaz e independente.
- Abertura: equipe scrum e stakeholders devem estar abertos sobre todo o trabalho e desafios inerentes ao trabalho.

A equipe Scrum (*Scrum Team)* é composta de:

- Scrum Master: tem o papel principal de organizar, garantir a execução dos ritos e remover impedimentos da equipe scrum.

- Product Owner: com o papel de definir e comunicar os objetivos do produto, criar, organizar e priorizar o Product Backlog e garantir que o Product Backlog seja transparente, visível e entendível.

- Desenvolvedores: com o papel de implementar os itens de backlog, mostrar o andamento, garantir o Definition of Done e ser adaptativo para atingir o objetivo das sprints.

A unidade básica do Scrum são as sprints. A sprint é um *time-box* onde os itens de backlog são priorizados para serem executados em um espaço de tempo fixo e definido. Cada sprint possui os seguintes ritos:

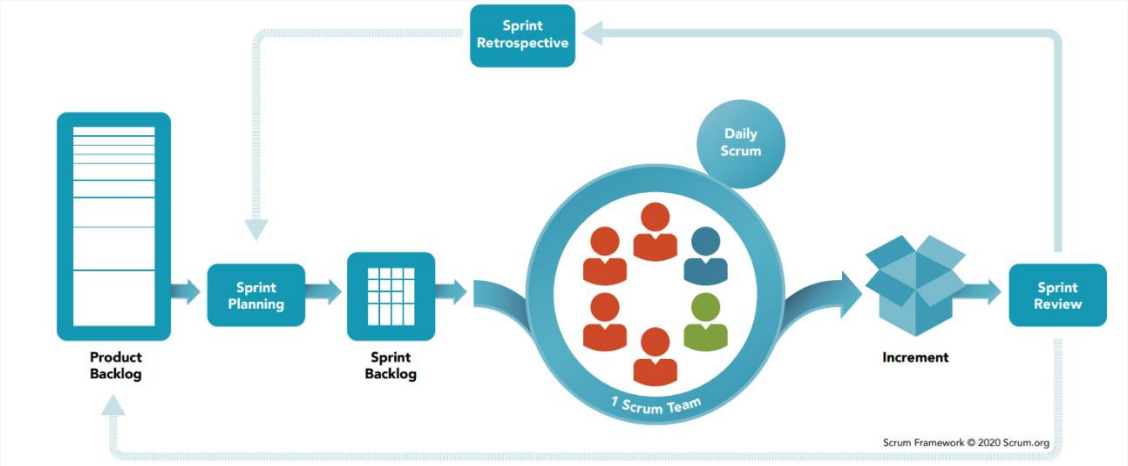
- Sprint Planning: é a reunião de planejamento e priorização do backlog que será entregue na sprint.

- Dailly: é uma reunião curta e diária com o objetivo que os membros da equipe relatem o que foi feito no dia, quais são os próximos passos e se possuem algum impedimento.

- Review: é a reunião executada ao fim da sprint para apresentar o que foi produzido durante a sprint.

Retro: é a reunião em que são discutidos os pontos positivos e a melhorar sobre a sprint que passou e levantar planos de ação para os itens a melhorar para serem executados na próxima sprint. Esta reunião é extremamente importante para garantir a evolução da equipe através do aprendizado contínuo.

Figura 21 – Scrum Framework.



Fonte: scrum.org.

O principal artefato do Scrum é o product backlog, que é basicamente uma lista de todas as atividades que devem ser executadas para incrementar o produto que o time está desenvolvendo. É importante ressaltar que o backlog é um artefato "vivo", ou seja, itens entram e saem o tempo todo assim como mudam de prioridade. Um backlog bem feito e bem priorizado é o coração de uma equipe Scrum de alto desempenho.

O Scrum hoje é o principal framework utilizado por times ágeis de desenvolvimento e é fundamental que os arquitetos de software coloquem seu dia a dia alinhado com as atividades das sprints.

Extreme Programming

O *Extreme Programming*, ou XP, também segue a mentalidade ágil no que diz respeito ao desenvolvimento de software e possui bastante semelhança com o Scrum no quesito de valores, principalmente. Os dois podem funcionar de forma complementar, uma vez que o Scrum gerencia melhor o fluxo gerencial do desenvolvimento e o XP a codificação em si.

O XP enfatiza trabalho em equipe. Gestores, clientes e desenvolvedores são iguais em uma equipe colaborativa e esta equipe deve

se auto organizar em torno do problema para resolvê-lo da forma mais eficiente possível.

O XP se baseia em cinco princípios essenciais:

- Comunicação: é fundamental que ocorra o tempo todo e que seja transparente entre os membros da equipe
- Simplicidade: os requisitos devem ser simples, pequenos desde o seu nascimento até a implementação e os testes
- Feedback: é o mecanismo pelo qual o time vai melhorar baseado nos retornos dados pelos clientes e time de desenvolvimento
- Coragem: esta é a coragem para dizer não na hora certa, ou então para dizer que o projeto vai demorar além do estimado, pois os novos requisitos precisam ser codificados e o código que já funciona precisa ser refatorado.

O XP também prega algumas boas práticas citadas abaixo:

- Cliente sempre disponível.
- *Planning Game.*
- Pequenas versões.
- Testes de aceitação.
- *Test First Design.*
- Integração contínua.
- Simplicidade de projeto.
- Refatoração constante.

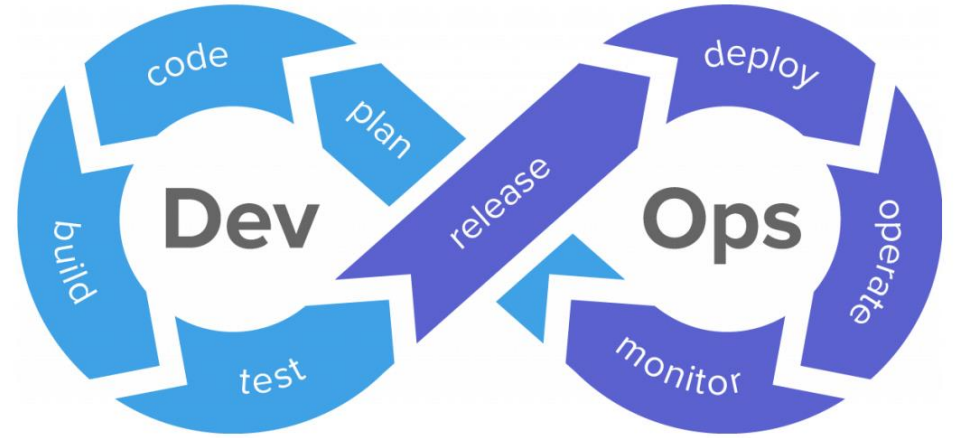
- *Pair Programming.*
- Rodízio de Pessoas.
- Propriedade coletiva – O código é de todos da equipe.
- Padronização de código.
- Jornada de 40 horas semanais.

Hoje o XP está sendo usado em larga escala em diversas partes do mundo nos mais diversos projetos de software, comprovando sua eficiência na entrega de produtos de software de qualidade.

Introdução ao DevOps

A palavra DevOps é simplesmente a união das palavras desenvolvimento e operações, porém seu real significado está muito além da simples união destas duas palavras. DevOps é um conjunto de boas práticas de colaboração com o objetivo único de entregar software de uma forma mais segura, ágil e escalável em ambiente de produção para gerar valor para um usuário.

Figura 22 – Ciclo Devops.



Fonte: https://www.dbacorp.com.br/devops/o-que-e-devops/.

Embora a adoção de práticas de DevOps automatize e otimize processos por meio da tecnologia, tudo começa com a cultura dentro da organização – e com as pessoas que fazem parte dela. O desafio de cultivar uma cultura de DevOps exige mudanças profundas na maneira como as pessoas trabalham e colaboram. Mas quando se comprometem com a cultura de DevOps, as organizações podem criar o ambiente ideal para o desenvolvimento de equipes de alto desempenho.

DevOps estimula várias práticas que ajudam esta engrenagem a girar da melhor forma:

- CI/CD (Integração e entregas contínuas).
- Controle de versão.
- Desenvolvimento ágil de software.
- Infraestrutura como código.
- Gerenciamento de configuração.
- Monitoramento contínuo.

É importante ressaltarmos que o ferramental técnico é essencial para que estas práticas sejam colocadas em ação. Existem diversas ferramentas na comunidade de software e devemos começar pela principal que será uma ferramenta de CI/CD. Para isso temos por exemplo o Azure Devops, Jenkins, Gitlab dentre outros.

A adoção de arquiteturas baseadas em cloud computing também está contribuindo fortemente como ferramenta de DevOps. A não-necessidade de gerenciamento de hardware físico auxilia muito na disponibilização de novas soluções de software em produção de forma mais ágil e escalável. As principais clouds públicas hoje são a Azure, Aws e Google Cloud, mas também existem outras que estão trabalhando para alcançar

estas como Oracle Cloud, IBM e mais recentemente a Cloud Flare também anunciou que investirá em soluções de nuvem.

A adoção de uma cultura DevOps dentro das organizações está se tornando cada vez mais estratégica para contribuir para aceleração do lançamento de novos produtos e contribuindo para a competitividade junto a um mercado cada vez mais dinâmico.

O papel da arquitetura de software no Devops e métodos ágeis

Neste momento é onde tudo se converge para um mesmo objetivo juntamente à arquitetura de software, entregar valor para a organização.

Quando estamos projetando uma arquitetura para um novo software devemos levar em consideração que ela deve se encaixar a boas práticas de DevOps e também devem ser viáveis de seguir métodos ágeis de desenvolvimento.

Ao definir os componentes de software que serão desenvolvidos eles devem ser flexíveis o suficiente para que consigamos priorizá-los dentro das iterações propostas pelo método de desenvolvimento e também estes componentes devem ser entregáveis através de uma esteira DevOps. Ainda agregando ao DevOps, nossa solução também deve ser capaz de ser monitorada e operada de forma simples e contínua em ambiente produtivo.

É papel do arquiteto pensar em uma solução que seja viável a toda esta integração desejada e mais importante ainda, que seja evolutiva dentro desse ambiente em constante mudança que as metodologias ágeis e o DevOps buscam se adequar.

> Capítulo 11

Capítulo 11. Gestão de Configuração e Versionamento

Gestão de configuração de software (GCS) é um conjunto de atividades de apoio que permite a absorção ordenada das mudanças inerentes ao desenvolvimento de software, mantendo a integridade e a estabilidade durante a evolução do projeto.

As atividades de Gestão de configuração são:

- Controlar e acompanhar mudanças (Controle de Mudança).
- Registrar a evolução do projeto (Controle de Versão).
- Estabelecer a integridade do sistema (Integração Contínua).

O controle de mudança abrange basicamente a gestão das mudanças que vão ser entregues através dos requisitos do software agrupados de acordo com a prioridade. Com isso conseguimos acompanhar, à medida que o desenvolvimento segue, o estado da solicitação da mudança até o lançamento de uma nova versão em produção.

Controlar a versão durante a evolução do projeto é basicamente o registro histórico das configurações nos artefatos do software, sendo que na maioria das vezes este artefato é o código. O controle de versão tem outras responsabilidades importantes: possibilitar a edição concorrente sobre os arquivos e a criação de variações no projeto.

Agora, a integração contínua fará o papel de garantir, a partir dos itens registrados em uma configuração, que ela será bem-sucedida. Aqui iremos fazer o build da aplicação através de scripts e ferramentas de automação que irão criar o artefato que será publicado e coletar métricas de qualidades relacionadas ao software que iremos entregar.

Versionamento utilizando Git

Git é o sistema de controle de versão mais utilizado no mundo atualmente. É um projeto open source criado em 2005 por Linus Torvalds, o criador do kernel do Linux. Com ele podemos criar todo histórico de alterações no código do nosso projeto e facilmente voltar para qualquer ponto para saber como o código estava naquela data.

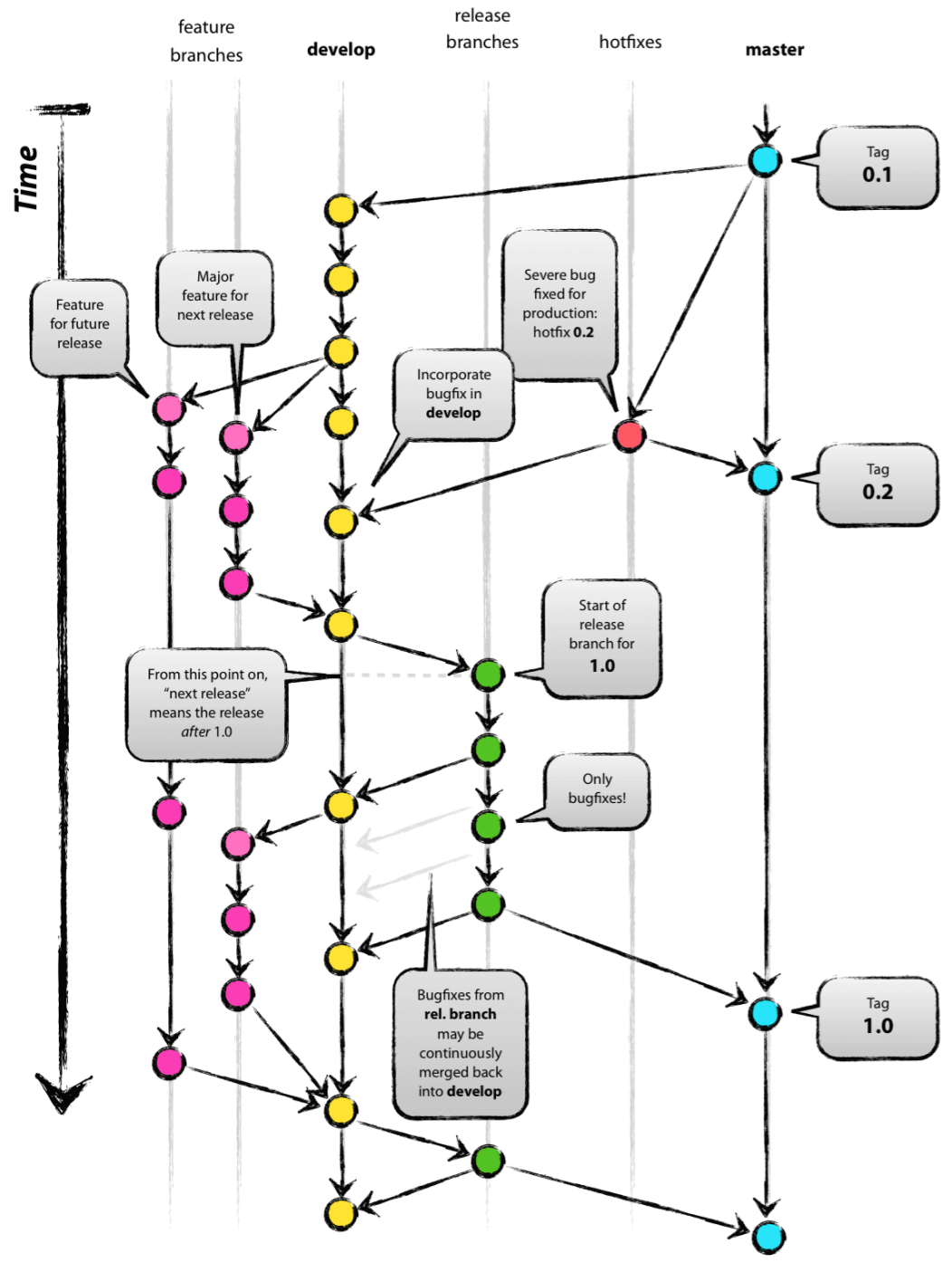
No git tudo é movido através dos commits, que consistem basicamente de um conjunto de alterações em um ou mais arquivos somados a um descritivo que resume as alterações nesse ponto.

O git também possui a possibilidade de criar ramificações ou branches que são formas de termos uma mesma versão do código sofrendo alterações e recebendo commits de diferentes fontes e diferentes desenvolvedores.

Gitflow

Gitflow é uma forma de fazer o gerenciamento das ramificações de um código através do git. Esta forma permite uma organização mínima das ramificações e um processo básico que pode ser adaptado de acordo com a necessidade do time de desenvolvimento. A figura abaixo representa como estes branches são organizados:

Figura 23 – Gitflow.



Fonte: https://nvie.com/posts/a-successful-git-branching-model/.

A seguir vamos explicar o papel de cada um destes branches:

- Master/Main: irá conter todo código já testado, versionado e que será entregue ao cliente. Esta branch deve sempre estar com a versão publicada em ambiente produtivo ou o mais próxima possível.

- Develop: sempre conterá o código mais atual, de onde as branches de feature são ramificadas tendo ela como base.

- Feature: novas implementações de novos recursos para o software são implementadas aqui.

- Release: também gerada a partir da branch de Develop, essa branch tem o papel de estabilizar um pacote a ser entregue na master, e consequentemente em produção.

- Hotfix: Para resolver problemas críticos em produção que não podem aguardar a entrega de uma nova release.

Versionamento semântico

> No mundo de gerenciamento de software existe algo terrível conhecido como inferno das dependências ("dependency hell"). Quanto mais o sistema cresce, e mais pacotes são adicionados a ele, maior será a possibilidade de, um dia, você encontrar-se neste poço de desespero. (Versionamento Semântico 2.0)

O versionamento semântico é um mecanismo que vem além do git para ajudar-nos a gerenciar as versões de pacotes e implementações que entregamos. É usualmente utilizado em pacotes que disponibilizamos em *feeds* de pacotes como NPM, pip e nuget, mas também pode ser facilmente aplicado a outros tipos de entregáveis, como rest api's ou aplicativos de celular.

Considerando uma versão 2.1.3 temos uma versão segmentada no formato *MAJOR.MINOR.PATCH* em que:

- Major: é incrementado quando criamos uma versão que gere uma incompatibilidade com as versões anteriores na API do pacote (breaking changes).

- Minor: é incrementado quando uma funcionalidade é adicionada mantendo compatibilidade com as versões anteriores.

- Patch: é incrementado quando lançamos versões com correções de falhas mantendo compatibilidade com versões anteriores.

Alguns rótulos adicionais podem ser adicionados no final com versões de pré-lançamento (-alpha, -beta) e metadados de construção (número do build). Exemplo: 2.1.3-alpha, 2.1.3.211008.

Referências

[1] BELLOQUIM, Atila. Framework de Arquitetura – Parte 2: TOGAF Disponível em: <https://arquiteturacorporativa.com.br/2010/09/frameworks-de-arquitetura-parte-2-togaf/>. Acesso em: 15 de jan. 2022.

[2] GROUP, The Ope. The TOGAF Standart, Version 9.2. Van Haren Publishing, 2018.

[3] CHEQUER, Gabriel Agostini. Alinhamento entre Arquitetura Empresarial e PDTI: Um estudo de Caso. Disponível em <https://www.maxwell.vrac.puc-rio.br/23138/23138.PDF> Acesso em: 15 de jan. 2022.

[4] BELLOQUIM, Atila. Framework de Arquitetura – Parte 1: Zachman. Disponível em: <https://arquiteturacorporativa.com.br/2010/06/frameworks-de-arquitetura-parte-1-zachman/>. Acesso em: 15 de jan. 2022.

[5] BRIDGES, Mark. Federal Enterprise Architeture Framework (FEAF): Business Reference Model (BRM). Disponível em: <https://flevy.com/blog/federal-enterprise-architecture-framework-feaf-business-reference-model-brm/>. Acesso em: 15 de jan. 2022.

[6] DA SILVA, Lana Montezano; RAMOS, Karoll Haussler Carneiro; FRANÇA, Joysse Vasconcelos; OLIVEIRA, Luisa Vilela Cury. Evolution of the enterprise architecture framework in the public administration of Brazil. Disponível em: <http://www.contecsi.tecsi.org/index.php/contecsi/16CONTECSI/paper/viewFile/6264/3644>. Acesso em: 15 de jan. 2022.