Desenvolvimento de Requisitos Arquiteturais

Bootcamp Arquiteto de Software

Augusto Farnese

2021

Desenvolvimento de Requisitos Arquiteturais

Bootcamp Arquiteto de Software

Augusto Farnese

Sumário

Capítulo 1. A Engenharia de Requisitos

Requisitos funcionais e não-funcionais

Requisitos funcionais dizem respeito a algo que o sistema deve fazer, como formatar um texto ou produzir um relatório. Em geral são as funcionalidades do sistema, ou aquilo que ele é capaz de executar. Um requisito funcional pode ter seu comportamento verificado por um conjunto finito de testes.

Em geral são levantados e especificados os requisitos funcionais do sistema. Os clientes pedem e descrevem o comportamento dos sistemas: o sistema deve fazer isso, não pode permitir aquilo etc. O conjunto de requisitos funcionais do sistema descreve a sua capacidade de resolver problemas. São requisitos mais tangíveis do que os requisitos não funcionais.

Já os requisitos não funcionais são aqueles que servem de restrições para a solução do sistema. Eles são mais abstratos do que os requisitos funcionais, pois muitas vezes não podem ser 100% verificados. Há várias categorias de requisitos não funcionais, e geralmente estão em uma das categorias:

1. **Desempenho** – Exemplo: o sistema deve ser capaz de aguentar muitos usuários simultâneos e responder em pouco tempo.
2. **Usabilidade** – Exemplo: os usuários devem conseguir realizar suas tarefas com rapidez e facilidade.
3. **Confiabilidade** – Exemplo: o sistema deve permitir desfazer todas as operações do usuário e garantir integridade dos dados
4. **Segurança** – Exemplo: ninguém deve ser capaz de acessar os dados de um usuário.
5. **Disponibilidade** – Exemplo: o sistema deve estar no ar 100% do tempo.
6. **Manutenção** – Exemplo: deve ser fácil dar manutenção no código do sistema.

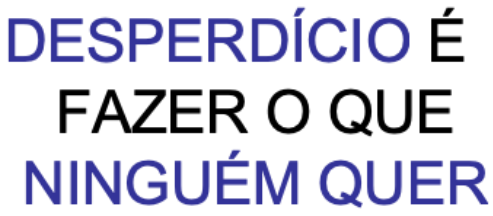
7. **Tecnologias envolvidas** – Exemplo: o sistema deve ser desenvolvido usando MEAN Stack.

É importante que esteja claro para as partes interessadas que é necessário priorizar os requisitos não-funcionais de um produto. Os clientes geralmente querem que o sistema contemple todos esses atributos de qualidade, mas isso pode tornar o projeto inviável. Para alcançar cada atributo não-funcional é necessário tempo e esforço da equipe de desenvolvimento. Muitas vezes isso não fica claro para os clientes (que acham que os requisitos não funcionais estavam implícitos) e ao final do projeto eles podem cobrar os requisitos não-funcionais e ficarem com suas expectativas quebradas.

Arquitetura Mínima Viável

O grande ponto do desenvolvimento Enxuto (Lean) é evitar desperdício. Ao elicitarmos os melhores requisitos e garantirmos uma priorização adequada, evitamos desperdício. Mas nem sempre é fácil realizar essas tarefas, em fato é sempre difícil.

Figura 1 – Desperdício é fazer o que ninguém quer.



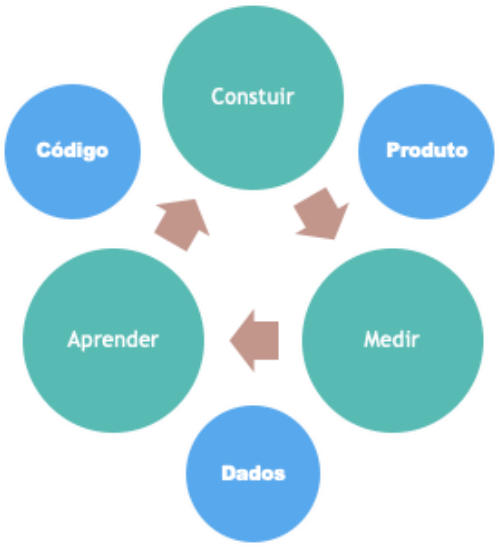
O conceito do desenvolvimento Enxuto veio de linhas de produção, mas também se aplicam bem ao desenvolvimento de software. Há vários sinais de desperdício em projetos de desenvolvimento, que podem ser identificados e evitados:

- Construir a funcionalidade errada.

- Má gestão do backlog.
- Retrabalho.
- Soluções desnecessariamente complexas.
- Excessiva carga cognitiva.
- Estresse psicológico.
- Espera / Multitask.
- Perda de conhecimento.
- Comunicação inefetiva.

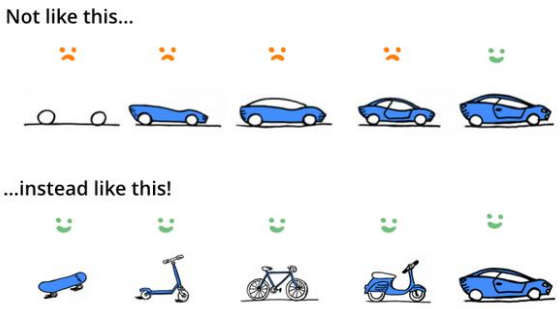
Uma prática para evitar o desperdício é estarmos sempre aprendendo sobre o contexto do produto e do projeto. Para isso é proposto um ciclo de aprendizagem que passa pelos itens *Construir* > *Medir* > *Aprender*. O objetivo é passar pelo ciclo muitas e muitas vezes, o mais rápido possível. Para isso construímos produtos pequenos que nos permitem ter feedback e aprender alguma coisa. Aprendizado é uma palavra de ordem no desenvolvimento enxuto.

Figura 2 – Ciclo de aprendizagem constante.



E para passar rapidamente pelo ciclo desenvolvemos produtos completos, no conceito de MVP.

Figura 3 – Queremos construir produtos que nos permitam rodar no ciclo e aprender.



Para a construção da arquitetura o mesmo pensamento deve estar presente na construção de uma arquitetura mínima viável. Queremos construir apenas o mínimo necessário para que possamos validar a nossa hipótese e dar suporte ao Sistema que estamos construindo na próxima iteração.

As atividades da Engenharia de Requisitos com desenvolvimento ágil são bastante diferentes de processo tradicionais. Estaremos trabalhando constantemente em comunicação com o time de desenvolvimento e em iterações curtas que permitem aprendizado.

Em processos tradicionais de desenvolvimento de software é produzido um artefato de especificação de requisitos, em que cada funcionalidade do sistema está rigorosamente descrita. Através desse documento, é possível fazer uma afiada verificação dos requisitos antes de começar a etapa de desenvolvimento.

Em metodologias ágeis a abordagem à especificação de requisitos é diferente. A negociação com o cliente é mais valorizada que a negociação de contratos, e, portanto, a especificação de requisitos deve ter como base a conversa e não a documentação.

As atividades que envolvem requisitos em processos ágeis de desenvolvimento de software devem estar alinhadas ao manifesto ágil. Em processos tradicionais todos os requisitos devem ser levantados, analisados e especificados anteriormente ao início da codificação. Nesses processos o documento que descreve os requisitos deve ser aceito (em alguns casos assinado) pelo cliente, de forma a garantir que não deverão sofrer alterações durante o projeto.

Em metodologias ágeis deve-se colaborar com o cliente para conseguir o software funcionando. Requisitos de software sofrem alterações o tempo todo, pois as regras que regem o funcionamento das organizações estão em constante mudança. Mudanças em leis acarretam mudanças nos requisitos. Em um projeto ágil é essencial estar ciente de tais riscos para que, se possível, responder às mudanças sem prejudicar o andamento do projeto.

Indivíduos e interações mais que processos e ferramentas;

Software em funcionamento mais que documentação abrangente;

Colaboração com o cliente mais que negociação de contratos;

Responder a mudanças mais que seguir um plano.

Há várias formas de se documentar requisitos. Em metodologias ágeis como XP e Scrum é bastante comum a utilização de Histórias de Usuários.

Figura 4 – O Manifesto Ágil.

12 princípios por trás do manifesto ágil

Uma forma de entender a atuação na Engenharia de Requisitos quando estamos trabalhando com processos ágeis é utilizar os “12 princípios por trás do manifesto ágil”:

1. Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.
    - Precisamos garantir a priorização de acordo com a geração de valor e não com quem grita mais ou quem tem mais poder.
2. Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam às mudanças, para que o cliente possa tirar vantagens competitivas.
    - Autodescritivo. ☺
3. Entregar software funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.

- Aprender com as entregas frequentes.

4. Pessoas relacionadas à negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.
    - Trabalhar juntamente com o time e com clientes
5. Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.
    - Trabalhar ainda mais junto com o time
6. O método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara.
    - Trabalhar abraçado com o time! ☺
7. Software funcional é a medida primária de progresso.
    - Valorizar entregas PRONTAS!
8. Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.
    - Fazer sua parte para promover o ambiente sustentável.
9. Contínua atenção à excelência técnica e bom design, aumenta a agilidade.
    - Ajudar o time a garantir a excelência técnica.
10. Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.
    - Priorizar, priorizar, priorizar...
11. As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis.

- Ser auto organizável com o time.

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

    - Aprender, aprender, aprender...

Engenharia de requisitos nos diferentes tipos de processos de desenvolvimento de software

As atividades do dia a dia de quem trabalha com requisitos pode ser bastante diferente de acordo com o tipo de processo utilizado no projeto de desenvolvimento de software. Alguns processos prezam por mais documentação e menos interação, enquanto outros vão valorizar mais interação a cara a cara e menos documentação extensiva. Esse e outros fatores podem influenciar muito as atividades de requisitos.

Um processo de desenvolvimento é a maneira que uma organização trabalha para produzir software. Há vários processos de desenvolvimento, com as mais variadas características. Os processos vão sempre passar por um ou mais estágios

Um processo definido tem documentação que detalha seus aspectos importantes: o que é feito, quando, por quem, quais recursos usa e produz. **Bons processos devem ajudar a produzir melhor, mais barato e mais rápido.**

Processo em Cascata

O modelo em cascata foi o primeiro modelo a se tornar amplamente conhecido e difundido na Engenharia de Software. Consiste em um modelo linear em que cada passo deve ser concluído antes do início do próximo. Há várias nomenclaturas para as etapas, mas basicamente temos algumas atividades que sempre estão presentes:

1. Requisitos.

2. Análise.
3. Desenho.
4. Implementação.
5. Testes.

A Figura 5 mostra uma representação gráfica das atividades básica de um modelo de processo em cascata. O Modelo Cascata é um modelo bastante simples e fácil de entender. Mas está em bastante desuso, pois houve muita inovação em termos de processos de desenvolvimento desde sua popularização. Em processos cascata as mudanças são muito caras, pois exigem que todas as etapas anteriores sejam refeitas.

Figura 5 – Modelo de ciclo de vida em cascata. Cada atividade começa ao final da anterior.

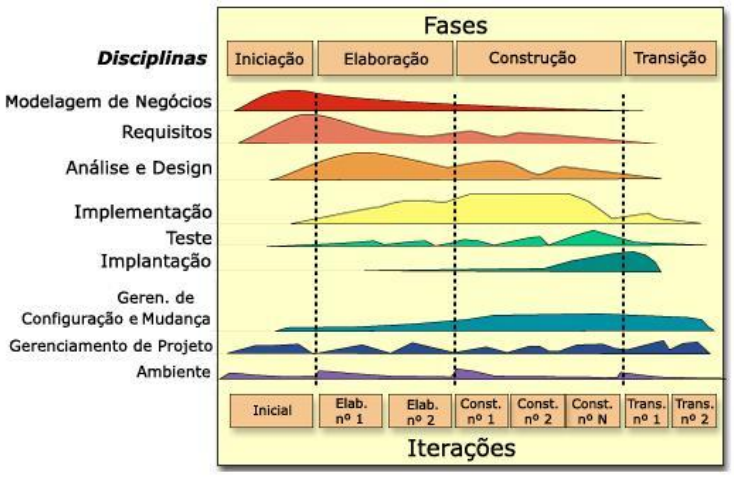
Em processos do tipo cascata as atividades de requisitos se concentram no início do projeto e depois são praticamente finalizadas, sem quase nenhuma atuação de quem faz o trabalho de requisitos. Isso leva a pouca interação com as outras equipes e a decisões preliminares no início do projeto.

Processos em Espiral

Processos em espiral trabalham de maneira mais iterativa do que processos em Cascata. É como se fossem a “Cascata Melhorada”. As atividades de requisitos

não são feitas TODAS no início do projeto, mas há uma grande concentração dessas atividades no início, como podemos ver no “gráfico de baleia” abaixo:

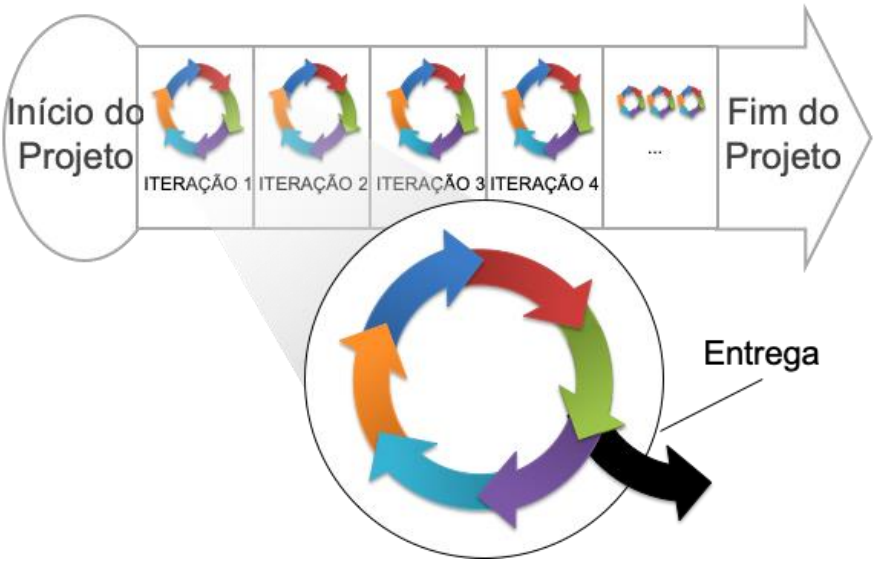
Figura 6 – Concentração das atividades em processos em espiral.



Processos ágeis (iterativos e incrementais)

Em processos iterativos e incrementais as atividades de requisitos estão distribuídas ao longo de todo o projeto. É como se todas as etapas do modelo em cascata fossem executadas a cada iteração, com potenciais entregas ao final.

Dessa forma, todas as etapas da Engenharia de Requisitos devem ser realizadas constantemente em cada iteração. A todo momento estamos elicitando, analisando, especificando e validando requisitos. Sempre com muita comunicação com a equipe de desenvolvimento.



As atividades de requisitos devem ser trabalhadas de maneira iterativa, sendo que o detalhamento dos requisitos aumenta a cada iteração. Os analistas de requisitos sofrem constante pressão da gerência de projetos para a definição dos requisitos de maneira geral, para que possam acompanhar a evolução do projeto de maneira linear. Entretanto, isso é uma ilusão que não deve estar presente em projetos que usam metodologias ágeis.

Os requisitos devem ser constantemente elicitados, analisados, especificados e validados, de maneira que cada ciclo de atividade alimente o próximo, para que as próximas atividades sejam mais bem realizadas. Os feedbacks colhidos na etapa de validação anterior serão muito importantes para as etapas seguintes, e assim sucessivamente.

O entendimento dos requisitos deve estar em contínua evolução, e o contato dos analistas de requisitos com as partes interessadas do projeto deve ser constante.

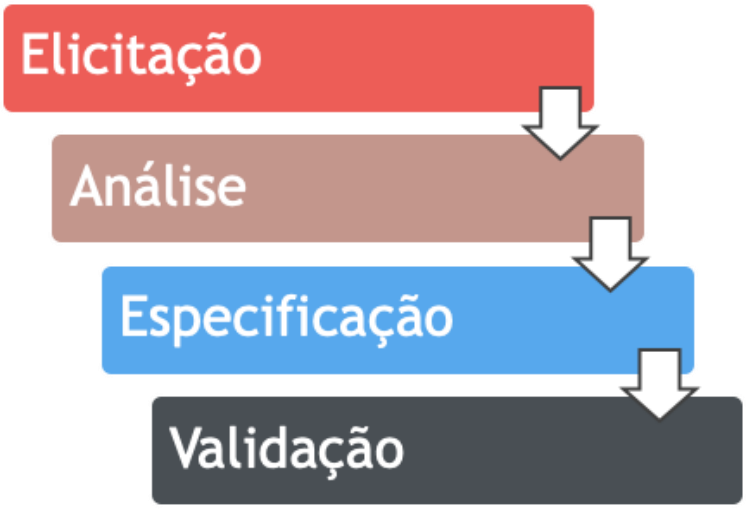
Capítulo 2. Elicitação De Requisitos Arquiteturais

Introdução à Engenharia de Requisitos

A Engenharia de Requisitos é uma subárea da Engenharia de Software que lida com todo o ciclo de vida dos requisitos de um projeto de desenvolvimento de software.

O processo da Engenharia de Requisitos pode ser dividido em quatro grandes atividades:

Figura 7 – Macro atividades da Engenharia de Requisitos.



Apesar das atividades fazerem sentido para todo tipo de projeto de desenvolvimento de software, a forma de execução das atividades do dia a dia pode variar muito de acordo com o tipo de processo utilizado (cascata, espiral ou iterativo) ou pelo tipo de produto (pode ser um produto encomendado ou um produto para o mercado).

Nas próximas sessões vamos detalhar os desafios de cada uma das etapas da Engenharia de Requisitos. Ao longo da disciplina e da apostila vamos falar de técnicas e abordagens para cada uma, sempre ligando aos desafios práticos da atuação.

Elicitação

A etapa de elicitação também pode ser chamada de “levantamento” ou “descoberta” de requisitos. Durante esta etapa há três grandes desafios:

- **Mapear as fontes de requisitos**: entender o contexto do problema e quais as melhores fontes de requisitos. Podemos levantar requisitos entrevistando pessoas, analisando concorrentes, estudando a legislação ou usando dados de utilização de outras ferramentas. Nesta etapa devemos planejar estrategicamente como e onde vamos descobrir esses requisitos.

- **Definir o escopo do projeto**: aqui precisamos deixar claro para as pessoas envolvidas no projeto o que será englobado pelo projeto, para não tentarmos “abraçar o mundo” e não conseguir gerenciar os requisitos. O alinhamento de expectativas deve ser bem feito para evitar frustrações futuras.

- **Conhecer o domínio do problema**: efetivamente realizar as atividades planejadas para entender os conceitos importantes do contexto do mundo real em que o software a ser desenvolvido irá atuar. Nesse momento realiza-se as entrevistas, faz-se as análises de dados etc. O grande objetivo é entender o problema que vai ser resolvido e conseguir ter contexto para a tomada de decisão de requisitos ao longo do projeto.

Análise

Uma vez que os requisitos foram devidamente elicitados, passamos para a etapa de análise desses requisitos.

Uma ferramenta para a análise é a modelagem conceitual dos requisitos. A geração de modelos pode ajudar nas discussões sobre os requisitos e servir de insumo para a definição da arquitetura da aplicação. Podem ser gerados modelos mais “formais” como diagramas UML, ou informais como desenhos simples.

Outro desafio da etapa de análise é a resolução de requisitos conflitantes. Por isso precisamos garantir a integridade dos requisitos e o atendimento às regras mais gerais previamente estabelecidas (normas internas, legislação etc.).

Priorização

Os requisitos devem ser priorizados de acordo com a geração de valor para o cliente. O conceito de valor varia muito de acordo com o contexto do projeto. O valor pode ser relacionado à redução de custo, ao aumento de vendas, à diminuição de reclamações etc. Isso deve estar claro para o analista de requisitos, pois é uma informação essencial para a etapa de priorização.

A prioridade dos requisitos ajuda a direcionar a ordem em que serão especificados e desenvolvidos. Em metodologias ágeis de desenvolvimento de software, as etapas de especificação e desenvolvimento acontecem de maneira incremental e evolutiva, de maneira que não é necessário que todos os requisitos estejam especificados para que sejam desenvolvidos. Por isso a priorização dos requisitos é uma tarefa essencial ao se trabalhar com processos ágeis.

Estabilidade dos requisitos

É importante que esteja claro o nível de instabilidade dos requisitos. Requisitos muito instáveis são aqueles que têm grande chance de mudarem ao longo do tempo. O analista de requisitos deve ter grande atenção aos requisitos instáveis, pois podem gerar retrabalho e prejudicar o sucesso do projeto e do produto.

Ao identificar requisitos instáveis, pode-se estabelecer uma arquitetura que seja mais maleável à mudança de requisitos, e reduzir um possível custo de mudança futuro.

Modelagem conceitual

A modelagem conceitual consiste na criação de modelos para facilitar o entendimento dos requisitos entre todas as partes interessadas. Tais modelos podem ser feitos em UML, BPMN, diagramas, casos de uso e diversas outras notações.

O principal objetivo da modelagem conceitual é ajudar a entender melhor o problema e a propor soluções melhores. Os modelos ajudam a abstrair questões irrelevantes do problema que podem ofuscar o caminho para uma solução.

Em alguns cenários o analista de requisitos deve criar diferentes modelos para um mesmo cenário, pois diferentes partes interessadas podem ter distintas visões de uma mesma situação. A equipe técnica pode querer um modelo mais técnico, enquanto a equipe de negócios pode querer um modelo mais processual. Há iniciativas como BPMN que têm o objetivo de unificar essas duas visões em uma única notação, mas nem sempre isso é possível.

A escolha da técnica e da notação para a modelagem conceitual vai depender da natureza do problema, do tipo de público e da experiência do analista de requisitos.

Na maioria dos projetos de desenvolvimento de sistemas de informação, a construção de modelos conceituais são uma excelente maneira de se trabalhar os requisitos e a definição do problema. Os modelos conceituais ajudam a estabelecer uma conexão entre o software a ser desenvolvido e o ambiente operacional.

Projeto arquitetural do sistema

Em muitas situações o analista de requisitos deverá trabalhar com o arquiteto de softwares. O detalhamento de alguns requisitos irá exigir conhecimento do impacto de determinada funcionalidade nos componentes e na estrutura arquitetural do sistema. Isso ajuda a priorizar e a ter uma estimativa em alto nível de cada requisito.

Negociação

A negociação de requisitos consiste na resolução de conflitos de requisitos. Após a etapa de levantamento de requisitos é bastante comum identificar requisitos conflitantes, resultado de opiniões distintas entre as partes interessadas ou simplesmente por conta de regras mal estabelecidas nas organizações.

Cabe ao analista de requisitos a negociação dos requisitos para resolver os conflitos. Muitas vezes exige habilidade de negociação para chegar a um alinhamento

entre os interessados ou simplesmente para garantir a definição e validação de alguma regra que não esteja estabelecida.

Especificação

Especificar os requisitos é registrá-los para que possam ser revisados e aprovados posteriormente.

Em processos tradicionais é gerada um extensivo documento de especificação de requisitos que contém todos os detalhes do software a ser gerado. Esse documento terá telas, detalhamento, regras, casos de uso etc. Geralmente é um documento utilizado posteriormente no desenvolvimento, feito para não ter muita necessidade de interação com a equipe que documentou os requisitos.

Já em processos ágeis, cria-se documentações mais enxutas. Uma ferramenta muito utilizada são histórias de usuários, uma maneira bastante simples de especificar os requisitos. O valor das Histórias de Usuários está na comunicação com o time e não na documentação em si. Mais adiante no curso falaremos sobre formas de documentar requisitos.

Validação

A etapa de validação consiste em garantir-se que o entendimento dos requisitos está alinhado às necessidades de clientes ou de quem usará a aplicação. Para isso podemos realizar apresentações, fazer testes com quem usará o sistema, trabalhar com protótipos em reuniões etc.

Também damos o nome de validação para o momento em que se verificar se o requisito realmente foi desenvolvido e consta na aplicação final. Nesse caso usamos testes de aceitação para fazer essa verificação.

Protótipos são uma ótima maneira de se validar os requisitos com os clientes. Ao visualizar um protótipo, as partes interessadas conseguem materializar seus

desejos e concentrar nas características importantes do software. Como software é um conceito abstrato, sem um protótipo pode-se perder o foco nas funcionalidades desejadas.

Tipos de conhecimento

O conhecimento do domínio do problema melhora a elicitação de conhecimento, principalmente em entrevistas. Nessas entrevistas é possível entender melhor o assunto e fazer perguntas melhores quando conhecemos melhor o domínio do problema.

Entretanto, uma expertise do domínio pode levar à omissão de conhecimento tácito, uma vez que ele passa a ser “trivial” para quem domina a área. “Conhecimento tácito é aquele que o indivíduo adquiriu ao longo da vida, pela experiência. Geralmente é difícil de ser formalizado ou explicado a outra pessoa, pois é subjetivo e inerente às habilidades de uma pessoa”.

"Um exemplo de conhecimento tácito é andar de bicicleta, pois trata-se de algo que é aprendido apenas a partir da experiência e tentativa, sendo desnecessário o uso de instruções escritas ou orais para aprender. "

Os grandes desafios ao lidar com conhecimento tácito na Elicitação de requisitos são:

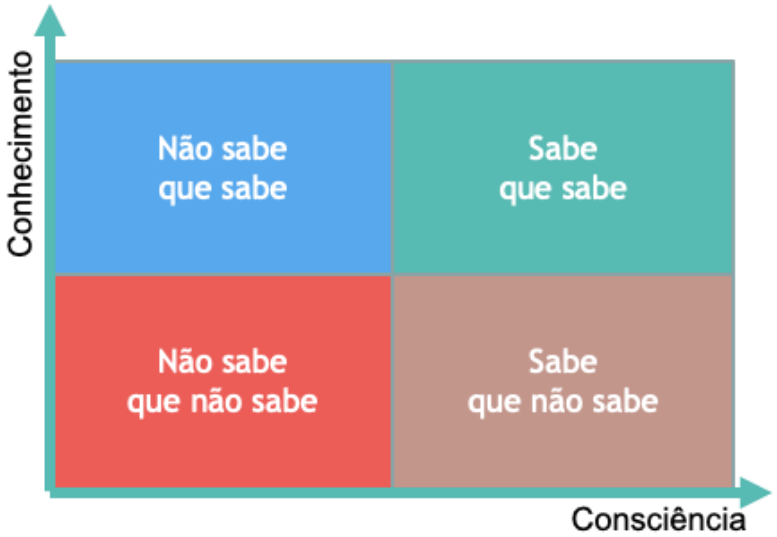
- Identificar.
- Saber o que é relevante e deve ser articulado.
- Articular o conhecimento no contexto certo para que seja conhecido por stakeholders.

Podemos dividir os tipos de conhecimento de acordo com o saber ou não e de acordo com a consciência ou não disso:

- Sabe que sabe.

- Sabe que não sabe.
- Não sabe que sabe.
- Não sabe que não sabe.

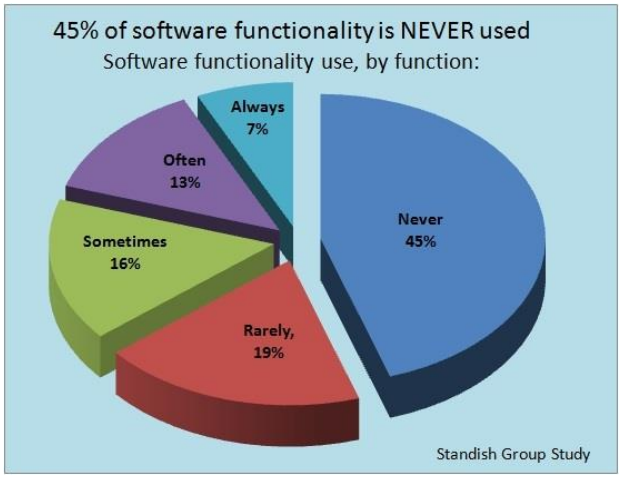
Figura 8 – Matriz de tipos de conhecimento.



Desafios da elicitação de requisitos

A Elicitação de requisitos é o primeiro ponto do trabalho com requisitos e tem grande impacto em todos os outros. Os requisitos podem definir o sucesso ou o fracasso de uma iniciativa de desenvolvimento de software. Um estudo mostrou que a grande maioria das funcionalidades de um produto nunca são usadas ou são raramente usadas (Figura 8). Isso mostra a importância da elicitação correta de requisitos para evitar desperdícios e fazer o que gera mais valor para clientes e usuários.

Figura 9 – CHAOS Report.



Neste mesmo relatório são analisados os fatores para o sucesso de um projeto, sendo considerado três características para definir sucesso: Custo, Tempo e Escopo. Ao longo dos anos os especialistas perceberam que essas características definem o sucesso do GERENCIAMENTO DO PROJETO e não do projeto. Então decidiram trocar o Escopo por SATISFAÇÃO. Mais do que fazer o que foi combinado, é importante alcançar a satisfação, o que só é possível ao garantir a geração de valor.

Lidar com os diversos perfis de pessoas.

Cada perfil de pessoa vai lidar com requisitos de uma maneira. Clientes, Analistas de Requisitos e Desenvolvedores têm visão diferente em relação aos requisitos.

- Cliente "Só mais uma funcionalidade, isso não deve ser difícil"
- Analista de requisitos "Isso não está claro, então vamos assumir que..."
- **Desenvolvedores "Aposto que eles também gostariam disso aqui..."**

Uma das responsabilidades da etapa de elicitação é identificar stakeholders do projeto:

- Têm posição relevante na empresa.
- Tomam as decisões no sistema novo.
- Dominam o problema.
- São expostas aos problemas perceptíveis.
- Podem influenciar a aceitação do sistema.
- Tem seus objetivos pessoais relacionados.

E lidar com os desafios do dia a dia:

- Recursos distribuídos e conflitantes.
- Dificuldade de acessar os recursos.
- Obstáculos para boa comunicação.
- Conhecimento tácito.
- Fatores sociopolíticos.
- Condições instáveis.

Capítulo 3. Análise de Requisitos Arquiteturais

Priorização de requisitos

Uma tarefa extremamente importante no trabalho de analistas de requisitos é a priorização para garantir a máxima geração de valor das atividades do time de desenvolvimento. Há duas ferramentas que podem ser úteis para a priorização:

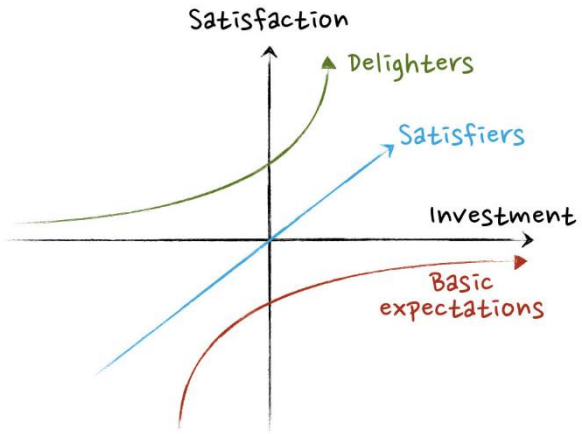
- Kano Model.
- Cost of Delay (Custo do Atraso).

Kano Model

Este modelo divide a funcionalidade de acordo com o comportamento delas quando analisamos um gráfico com dois eixos. No eixo vertical analisamos a satisfação dos usuários com uma determinada funcionalidades. No eixo horizontal analisamos o investimento que é feito na funcionalidade, resultando em algo melhor ou pior implementado. Temos então três categorias de funcionalidades:

1. Expectativas básicas (basic expectations): são funcionalidades básicas que necessariamente devem estar presentes e bem implementadas. Caso esteja mal implementada vão ter um impacto muito grande na satisfação dos usuários.
2. Satisfação (Satisfiers): são funcionalidades com um comportamento linear entre investimento e satisfação. Se estiverem bem implementadas resultarão em grande satisfação e se estiverem mal implementadas terão péssimo impacto na satisfação.
3. Encantamento (Delighters): são funcionalidades que se estiverem ausentes não terão impacto na satisfação dos usuários, mas se estiverem presentes a satisfação com aquela funcionalidade será muito alta.

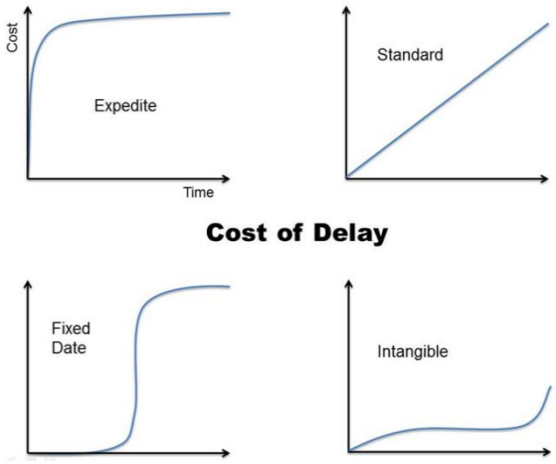
Figura 10 – Gráfico do Kano Model com os tipos de funcionalidades



Cost of Delay (Custo do Atraso)

Na técnica de Cost of Delay consideramos o custo que uma organização tem ao "não ter" uma funcionalidade. Descobrimos "quanto custa ficar sem uma funcionalidade", para poder estimar o custo do atraso. Há alguns comportamentos em relação ao tipo de funcionalidade, como podemos ver na Figura 12.

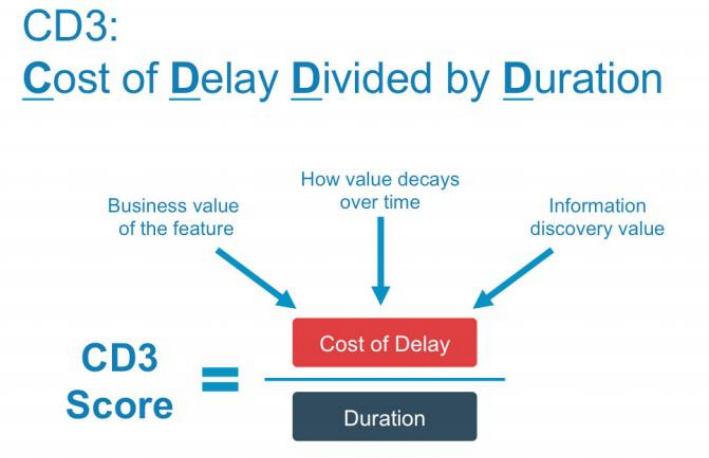
Figura 11 – Evolução do custo do atraso de acordo com o tipo de demanda.



Para calcular um índice de priorização de acordo com o atraso, dividimos o custo do atraso pela duração da funcionalidade (Figura 11). Quanto maior o resultado,

mais prioritária deve ser uma tarefa. Dessa forma ligamos a priorização diretamente aos custos financeiros.

Figura 12 – Índice de custo do atraso dividido pela duração.



Na Figura 13 podemos visualizar um exemplo de cálculo com três funcionalidades com custos de atraso e duração diferentes.

Figura 13 – Exemplos de cálculo de custo do atraso usando o CD3.

	CoD por semana	Duração Planejada	Score
Funcionalidade A	R$ 2.000	5 semanas	400
Funcionalidade B	R$ 30.000	6 semanas	5.000
Funcionalidade C	R$ 8.000	2 semanas	4.000

Escrevendo Histórias de Usuários

Uma história de usuário é, essencialmente, uma breve descrição de um requisito do ponto de vista de um usuário do sistema. A descrição é feita como se o usuário estivesse descrevendo uma de suas necessidades e justificando-a. Um exemplo de história de usuário para um sistema de gestão de projetos seria:

Eu, gerente
preciso acompanhar as atividades de minha equipe

porque preciso garantir que as atividades sejam executadas dentro do prazo.

As histórias de usuários têm como objetivo promover a conversa entre a equipe. Ao contrário de uma especificação formal de requisitos, ela é incompleta por definição. Dessa forma é essencial que a equipe de desenvolvimento esteja em contato constante com o Dono do Produto para que as dúvidas e os pontos em aberto sejam resolvidos e esclarecidos.

Para entender a forma incompleta das histórias de usuário, usamos a analogia da foto. Se você vê uma foto de um amigo na praia tomando uma água de coco, vai pensar apenas “Aqui está meu amigo na praia tomando água de coco”. Agora imagine uma outra situação, em que seu amigo te conta uma história dessa foto: ele conta como ele teve dificuldades para chegar na praia, diz que torceu seu pé no caminho até lá e que estava com tanta sede que teve que pagar R$50,00 naquele coco e que nem achou tão gostoso. Quando você olhar novamente para a foto não vai mais pensar apenas “Aqui está meu amigo na praia tomando água de coco”. Dessa vez você vai se lembrar de cada detalhe que ele te contou, vai pensar no quão caro foi aquele coco e pensar como o pé do seu amigo estaria doendo.

Nos dois casos temos a mesma foto, mas com a conversa a foto passa a ter muito mais valor e conteúdo. Da mesma forma funcionam as histórias de usuários. Quando vemos uma história de usuário ela está incompleta, mas quando conversamos com o Dono do Produto para entender as necessidades dos clientes tudo se torna mais claro. A partir da conversa a história tem mais valor. O time de desenvolvimento entende o problema e passa a enxergá-lo do ponto de vista do usuário. Trabalha-se então para resolver o problema do usuário.

As histórias de usuário são simples, mas é necessário bastante experiência para que se escreva boas histórias e para explicá-las da melhor forma possível para toda a equipe. O objetivo é o alinhamento de conhecimento sobre o problema, e para isso a conversa constante e abundante é essencial. A disponibilidade do Dono do Produto para o time de desenvolvimento deve ser constante.

Débitos técnico

Ao longo das atividades de desenvolvimento, o time se depara com situações em que se decide não usar as melhores práticas de codificação. Isso pode ser por questão do prazo de uma entrega, pela instabilidade dos requisitos etc. Quando isso acontece, um débito técnico é colocado no produto.

É importante gerenciar o débito técnico para que se garanta que os problemas serão resolvidos com o tempo, pois sem a devida gestão cada vez mais problemas são colocados e não são resolvidos. A longo prazo, o produto aumentará de complexidade e sua manutenção fica cada vez mais custosa.

Com a devida gestão o débito técnico é acompanhado e atividades de refatoração são realizadas para corrigir os problemas mais críticos.

Acessibilidade

Acessibilidade é a possibilidade de acesso a um lugar ou conjunto de lugar. No caso da web, uma interface com boa acessibilidade provê boa experiência de navegação para qualquer pessoa, independentemente de terem alguma deficiência ou não.

Problemas de sites não acessíveis

Há diversos tipos de deficiência que podem atrapalhar a utilização de um site não acessível. A tabela abaixo relaciona algumas dessas deficiências e os potenciais problemas que podem ter:

Deficiência	Exemplos de problemas em sites não acessíveis
Cegueira	Sem utilizar corretamente as semânticas HTML, descrição de imagens ou links bem nomeados, é difícil, até impossível, navegar no site com leitores de texto.

Daltonismo	Links diferenciados apenas pela cor podem não ser identificados por usuários daltônicos.
Problemas motores	Links e áreas clicáveis pequenas podem ser difíceis de acessar por usuários que não podem controlar o mouse com precisão.
Dislexia e dificuldades de aprendizagem	Conteúdo complexo e sem imagens instrucionais podem dificuldade a compreensão do conteúdo.
Deficiências de visão	Textos e imagens pequenas, e sem possibilidade de ampliação, dificultam a leitura e o entendimento por usuários com problemas de visão.

Acessibilidade também diz respeito às recomendações do W3C, para sites acessíveis. Existem diversos validadores disponíveis para verificar automaticamente se o site está aderente às recomendações da W3C.

O governo federal criou o eMAG – Modelo de Acessibilidade em Governo Eletrônico. Todos os sites e portais do governo brasileiro têm obrigação de implementá-lo. Disponível em: http://www.governoeletronico.gov.br/acoes-e-projetos/e-MAG

Por que investir em acessibilidade

Estudos mostram que cerca de 20% da população brasileira possui algum tipo de deficiência. Certamente nem todas essas pessoas possuem incapacidades que as impeçam de utilizar um site não acessível. Mesmo que o percentual de usuário incapazes de utilizar um site não acessível seja 10% ou até 5%, é imprudente para qualquer negócio excluir tal percentual de potenciais clientes.

Para uma loja on-line, por exemplo, alcançar 5% a mais de público pode representar mais vendas. Para uma rede social, representa mais usuários. Projetar um site acessível, então, é um investimento que deve ser considerado em projetos web.

Desempenho

As tecnologias de desenvolvimento web tem evoluído muito nos últimos anos. Com isso, estamos tentando fazer mais e mais coisas em nossas aplicações. Consequentemente, as aplicações estão ficando mais robustas e mais pesadas, podendo levar um longo tempo de carregamento e demora para os usuários.

Com isso um problema tem ficado muito evidente: desempenho. Quanto mais tentamos fazer sistemas robustos, capazes de fazer muitas coisas, mais o desempenho se torna uma questão relevante para desenvolvedores.

Existe um objetivo primário para se preocupar com desempenho: dar ao nosso usuário uma experiência melhor. Empresas conseguem aumentar consideravelmente a retenção de clientes e a rentabilidade de seus sites ao melhorar o desempenho da aplicação e, consequentemente, a experiência de seus usuários.

Aplicações são essenciais para cada vez mais atividades fundamentais para as pessoas e o desempenho das aplicações, que não deve ser uma barreira para que elas consigam alcançar seus objetivos.

Veremos várias formas de melhorar o desempenho das aplicações, mas você não precisa se preocupar em usar todas. Qualquer coisa que você puder fazer para melhorar o desempenho já poderá afetar positivamente a experiência do usuário.

Os problemas de desempenho se concentram em dois pontos principais da aplicação: no carregamento da página e na renderização na página. O carregamento da página é quando todos os recursos estáticos são carregados no servidor para o navegador. Isso significa que quanto maiores forem os arquivos, maior será o tempo de carregamento. O tempo de renderização diz respeito ao tempo que demora depois que todos os arquivos essenciais foram carregados, e o navegador vai precisar trabalhar para compilar, interpretar tudo e dispor os elementos de uma maneira que seja entendível visualmente.

Devemos tratar a questão de desempenho como uma questão estratégica para nossa aplicação. Vários estudos mostram que aplicações com melhor

desempenho tem melhores resultados. Um site de vendas por exemplo, poderá perder usuários ou perder vendas por conta de demora na resposta da aplicação. O usuário poderá fazer o cadastro no sistema porque aplicação demora muito, e ele acabou desistindo. Então devemos tratar desempenho como atributo diretamente ligado aos objetivos de negócio, por isso devemos decidir estrategicamente quanto tempo e esforço vamos investir para melhorar o desempenho da aplicação.

Usabilidade

Usabilidade define a facilidade com que as pessoas podem empregar uma ferramenta para realizar uma tarefa específica e importante. Dessa forma, a interface web deve ser construída para ajudar o usuário a realizar seu trabalho e chegar à informação importante de maneira efetiva. De acordo com a ISO 9241-11: “Usabilidade é a medida pela qual um produto pode ser usado por usuários específicos para alcançar objetivos específicos com efetividade, eficiência e satisfação em um contexto de uso específico.”

Atributos de usabilidade

Para garantir alto grau de usabilidade, as interfaces do sistema devem contemplar os cinco atributos de usabilidade listados abaixo:

Aprendizado	•A interface deve ajudar o usuário a aprender como usar o sistema.
Eficiência	•Quando o usuário estiver experiente na utilização do sistema, ele deve conseguir fazer suas tarefas rapidamente.
Memorização	•Mesmo depois de um longo período sem usar o sistema, o usuário deve rapidamente reestabelecer seu nível de proficiência.
Robustez	•Quando o usuário comete algum erro o sistema deve facilmente ajudá-lo a se recuperar desse erro, sem prejudicar a utilização.
Satisfação	•Utilizar o sistema tem que ser agradável. O usuário precisa ficar satisfeito ao utilizá-lo.

Exemplo: sistema bancário

Um sistema bancário é um exemplo de sistema de difícil aprendizado, mas muito eficiente. Um funcionário do banco deve ter vários treinamentos para entender os conceitos e as funcionalidades do sistema. Um usuário leigo certamente não conseguiria utilizar um software bancário sem treinamento.

Entretanto, quando o usuário está familiarizado com o sistema ele consegue chegar a seu objetivo facilmente. Utilizando apenas o teclado, com poucos comandos o funcionário do banco consegue resolver o problema de um cliente. Apesar de ser de difícil aprendizado, é bastante eficiente.

Capítulo 4. Especificação e Validação de Requisitos Arquiteturais

Casos de uso

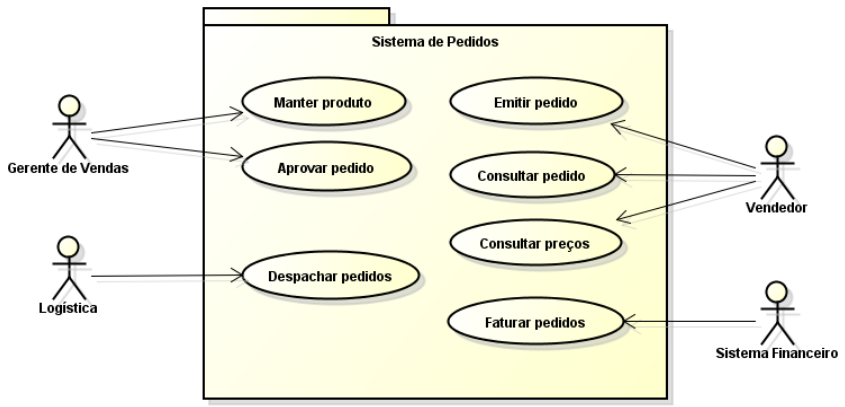
Casos de uso são uma forma de se representar requisitos funcionais de uma aplicação. Cada Caso de Uso é uma unidade de trabalho funcional realizado por uma pessoa no sistema.

É uma ferramenta amplamente utilizada em projetos de desenvolvimento de software e já tem sua efetividade mais que comprovada. Os Casos de Uso são descritos de maneira textual, o que permite que seja facilmente compreendido pelas diversas partes interessadas de um projeto. Assim, analistas de requisitos, clientes, *stakeholders* e o time de desenvolvimento podem trabalhar e discutir em uma mesma representação dos requisitos.

Diagramas de Casos de Uso

Uma forma visual de representar os Casos de Uso são os diagramas de Casos de Uso. É um tipo de diagrama da UML que permite verificar a interação entre atores do sistema, casos de uso e entre casos de uso. Geralmente utiliza-se uma fronteira para identificar em qual sistema, ou parte do sistema, os casos de uso se aplicam. A Figura 14 mostra um exemplo de diagrama de casos de uso contendo vários atores e vários casos de uso.

Figura 14 – Diagrama de Casos de Uso.



Estrutura de um Caso de Uso

Como um Caso de Uso representa uma unidade funcional do sistema, ele deve conter todas as informações necessárias para o entendimento daquele requisito. Claro que é necessário também um contexto mais amplo para o entendimento, pois os Casos de Uso não são autocontidos.

As principais informações que devem conter em um Caso de Uso são:

- **Nome**: um identificador daquele caso de uso.
- **Sumário**: breve descrição do comportamento daquele caso de uso.
- **Pré-condições**: condições específicas que devem ser atendidas para que aquele caso de uso faça sentido e para que possa ser executado. Geralmente estamos falando de dados que devem conter no sistema, ou situações específicas que devem ser atendidas.
- **Gatilhos**: o que deve acontecer no sistema para que o caso de uso seja "ativado". Pode ser o clique em um botão, uma ação automática do sistema ou qualquer outro evento que faça com que o Caso de Uso seja executado.
- **Linha de Eventos**: quais são os eventos que acontecem quando o caso de uso é executado.
- **Percursos Alternativos**: além do fluxo feliz, em que tudo dá certo e corre conforme o planejado, quais são os fluxos alternativos que podem acontecer? Aqui podemos considerar fluxos com ou sem incidência de erros.
- **Pós condições**: quais condições devemos esperar após a execução de um caso de uso.
- **Regras de negócio**: quais regras e restrições de negócio se aplicam para a execução deste caso de uso?

Há várias vantagens de se utilizar casos de uso, como:

- São fáceis de entender e servem como ponte entre quem desenvolve, quem usa, clientes e partes interessadas no projeto.
- Permite a visualização de caminhos alternativos do fluxo, além daquele que esperamos.
- Estão diretamente ligados à interação com o sistema.
- Há uma notação padronizada para a criação de diagramas de Casos de Uso, o que facilita a documentação e manutenção dos requisitos.

Também podemos listar alguns riscos ou desvantagens da utilização de casos de uso:

- Podem não ser apropriados para a representação de requisitos não-funcionais.
- A utilização de templates pode passar a sensação de prover clareza, mas isso não é verdade. Vai depender da habilidade de quem for escrever os casos de uso para garantir que sejam claros, sucintos e apropriados para a discussão dos requisitos.
- Os Casos de Uso não são autocontidos e precisam de um contexto maior para que sejam compreendidos. Eles descrevem bem a funcionalidade, mas não representam o contexto de uso do projeto e do produto.
- O uso extensivo de Casos de Uso pode tornar a equipe muito centrada em documentação e acabar evitando discussões e comunicação constante. Isso deve ser um ponto de atenção para quem utiliza Casos de Uso em cenários ágeis.

Casos de Uso e Histórias de Usuários

Quando comparamos casos de uso com histórias de usuários, temos algumas diferenças:

- Casos de uso são focados na solução proposta, enquanto Histórias de Usuários são mais centradas no problema que uma determinada pessoa tem no mundo real e que vamos resolver com software.

- Casos de Uso podem ser mais centrados na documentação enquanto Histórias de Usuário vão enfatizar a comunicação face a face.

- Para processos mais tradicionais os Casos de Uso se encaixam melhor. Para processos ágeis é mais comum a utilização de Histórias de Usuários.

É importante ressaltar que não existe uma ferramenta que seja melhor que a outra. Ambas têm pontos fortes e fracos e é de responsabilidade da equipe do projeto decidir qual a ferramenta mais apropriada para documentar os requisitos.

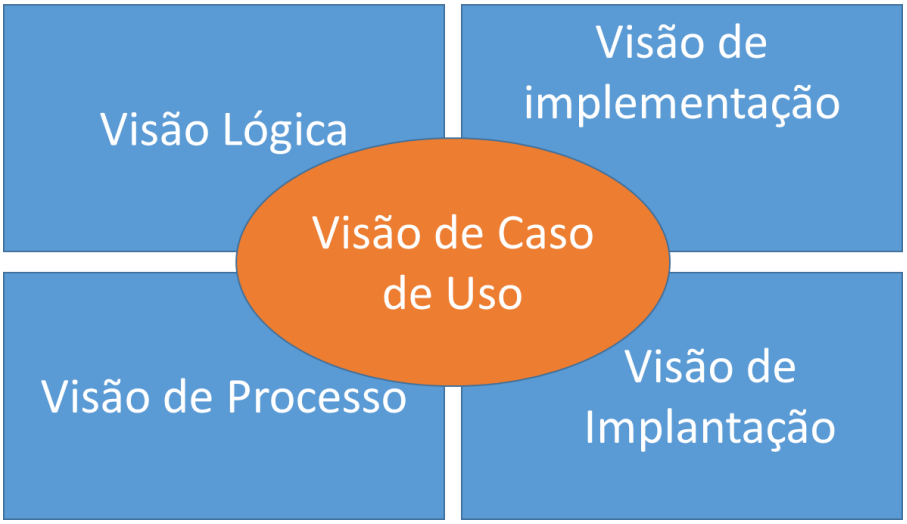
Modelo 4+1 de Visão Arquitetural

Um modelo é uma abstração da realidade. Fazemos modelos para capturar os elementos mais importantes para termos uma boa discussão e tomarmos decisões para os projetos e produtos. Fazemos isso pois seria difícil discutir considerando TODOS os aspectos envolvidos.

Usar modelos nos ajuda a entender os problemas e estruturar as soluções, experimentar possíveis alternativas, simplificar questões complexas, identificar riscos e, com isso, evitar erros.

Apesar de ser importante trabalhar com modelos, precisamos organizá-los para que possamos ter a visão que queremos do sistema, mas que não se tornem extremamente complexos a ponto de não ser possível usá-los.

Pensando nisso, foi proposto o Modelo 4+1 de visões arquiteturais:



Assim é possível organizar os modelos, tratá-los de acordo com o tipo de visão e decisões associadas para trabalhar em alto nível de abstração.

UML

UML é uma sigla para Linguagem de Modelagem Unificada (Unified Modeling Language). É uma notação gráfica para representar sistemas e interações. A UML define diagramas padronizados e tem complexidade suficiente para modelar quase qualquer aspecto de um sistema.

A ideia de unificação é porque combina conceitos de várias notações e tem uma abrangência grande para modelagem de negócios, detalhamento de requisitos, análises, desenho de sistemas e até implementação, testes e implantação.

UML é independente de linguagem, plataforma ou processo e é suportada por diversas ferramentas para fazer a modelagem.

Há vários diagramas UML, que são divididos em Diagramas de Estruturas e Diagramas de Comportamentos.

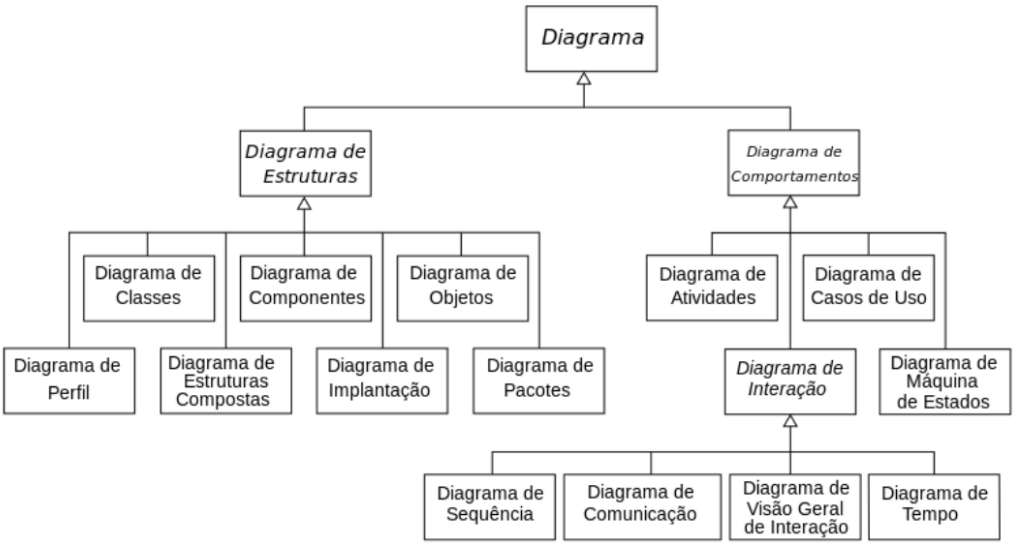
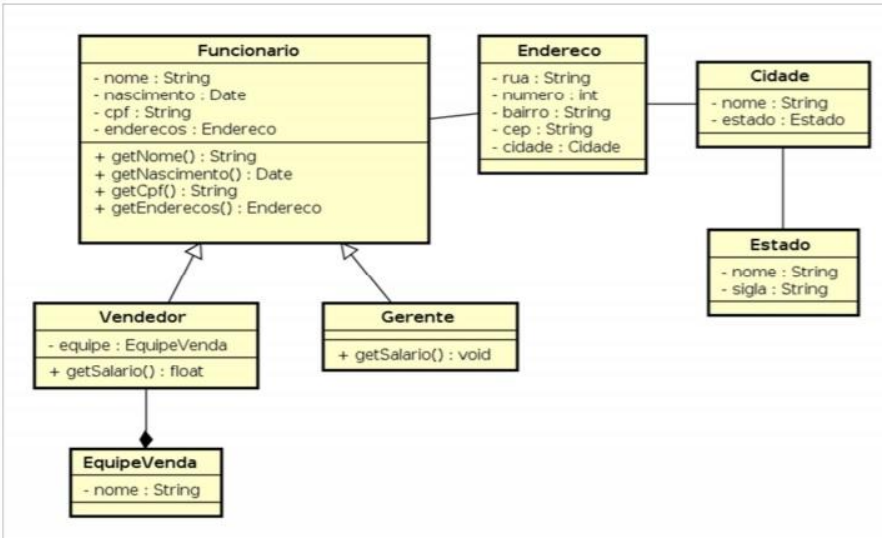


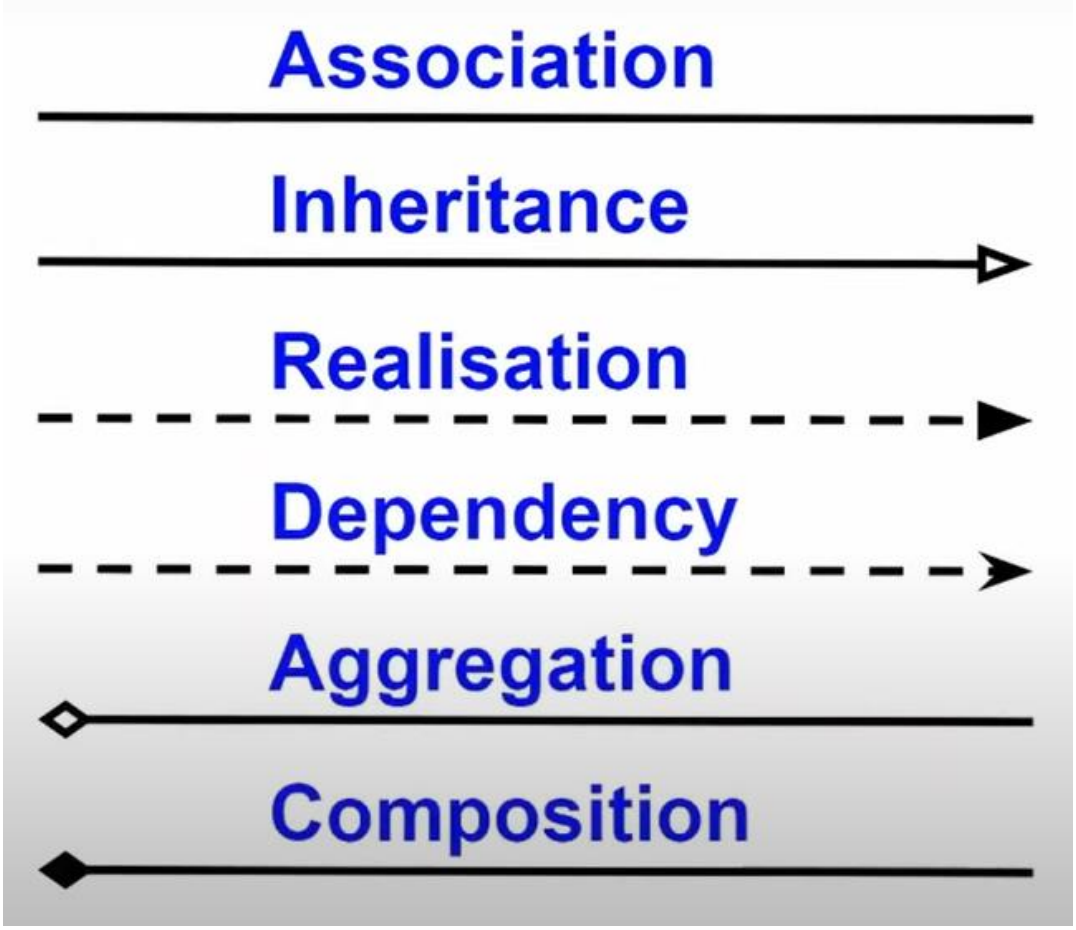
Diagrama de Classes

O diagrama de classes ilustra a estrutura de um sistema descrevendo as Classes e seus atributos, métodos e relacionamentos. O Diagrama de Classes é o mais importante e o mais utilizado diagrama da UML, e serve de apoio para a maioria dos outros diagramas.



Uma Classe é um modelo para a instanciação de um Objeto. O Objeto instanciado vai ter suas características próprias e atributos. A Classe apenas descreve a estrutura desse Objeto.

Há vários tipos de possíveis relacionamentos entre classes:

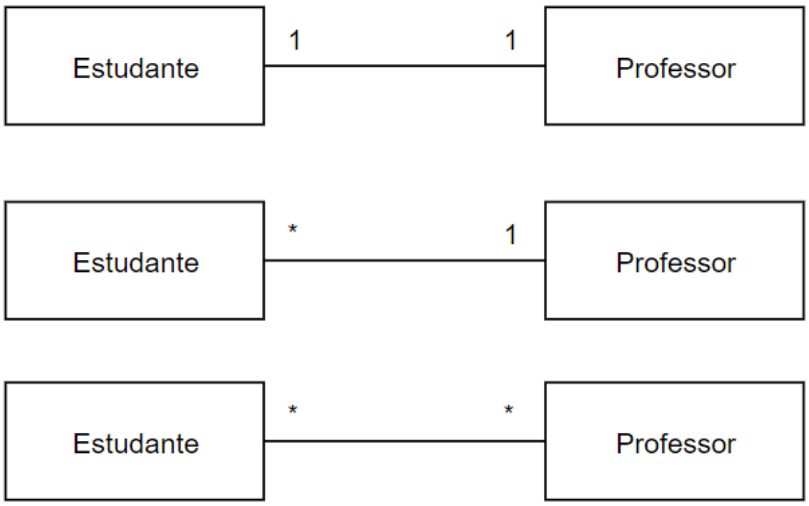


Associação

Descreve um vínculo simples entre duas classes e determina que as instâncias de uma classe estão de alguma forma ligada às instâncias da outra classe.

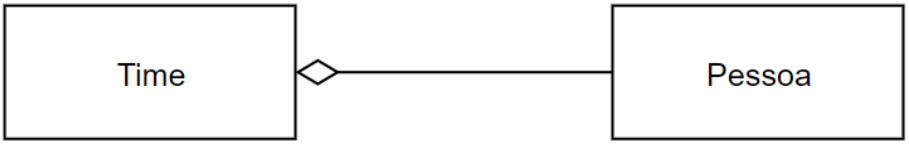
Multiplicidade

Todas as relações entre classes podem ter multiplicidade, indicando quantas instâncias de uma classe estarão vinculadas à outra.



Agregação

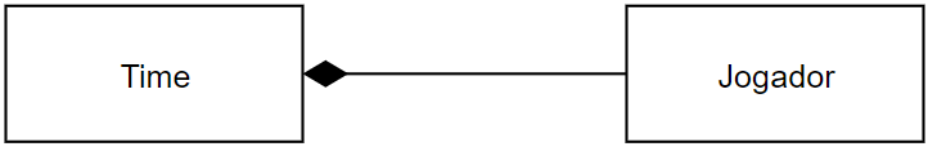
Agregação é um tipo especial de associação. Demonstra que as informações de um objeto precisam ser complementadas por um objeto de outra classe.



No exemplo acima, um time só faz sentido se tiver pessoas. Não dá para passar as informações de um time se não houver pessoas vinculadas a ele.

Composição

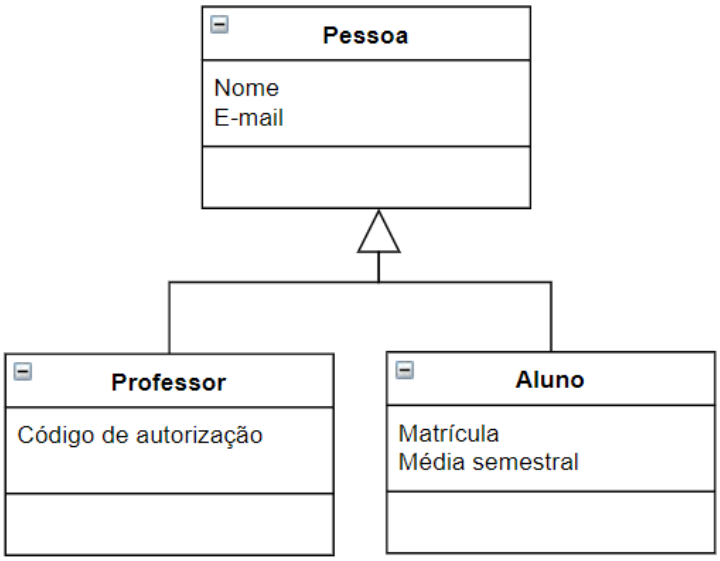
É um vínculo ainda mais forte entre objetos-todo e objetos-parte. Eles precisam estar vinculados. O todo não faz sentido sem as partes, ou as partes não fazem sentido sem o todo.



No exemplo acima (que muda “Pessoa” para “Jogador” em relação ao anterior) já temos uma composição. Uma classe Jogador não vai existir se não existir um time.

Especialização / Herança

Indica quando uma classe é uma especialização de outra classe, sendo a relação do tipo “é um”. Tudo que as classes gerais podem fazer as classes especializadas também podem, tendo todos os atributos e métodos herdados. A classe “filha” pode ter mais atributos ou métodos específicos.



No exemplo acima todo professor ou aluno é uma pessoa, mas com características e atributos específicos.

Diagrama de Sequência

É um diagrama UML que representa a forma como os objetos interagem para executar um serviço específico. O diagrama captura o comportamento de uma funcionalidade (ex.: um caso de uso específico), sendo a interação representada através da troca de mensagens.

O diagrama é feito levando em consideração a ordem temporal em que as mensagens são trocadas. Deve-se identificar no diagrama o evento gerador da funcionalidade modelada e os objetos envolvidos na ação.

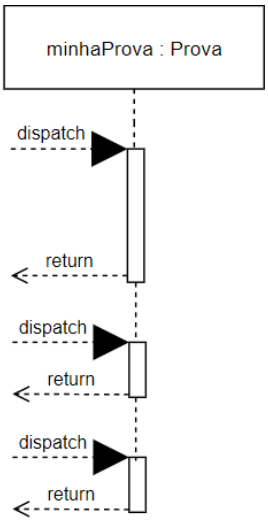
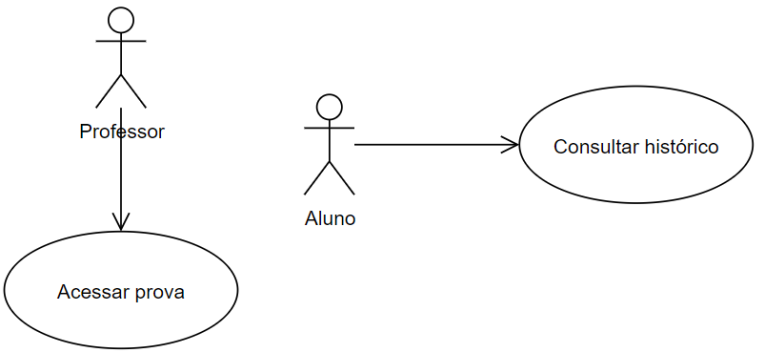


Diagrama de Casos de Uso

Um diagrama com linguagem simples, que também é acessível para clientes (em geral os outros diagramas são mais complexos, sendo usados e entendidos apenas por desenvolvedores). O diagrama serve para se entender o comportamento externo do sistema por partes interessadas, e apresenta o sistema da perspectiva do usuário.

É o diagrama mais abstrato da UML (tem poucos detalhes) e por isso é o mais flexível e informal. É geralmente usado no início da modelagem do sistema, muito usado em especificações de requisitos.

É formado por atores, casos de uso e relacionamento entre eles.



Referências

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML, Guia do Usuário.** 2. ed., Editora Campus, 2005.

FILHO, Wilson de Pádua Paula. **Engenharia de software**. Vol. 2. LTC, 2003.

FOWLER, Martin. **UML Essencial**. 2. ed. Bookmann, 2000.

PATTON, Jeff; ECONOMY, Peter. **User story mapping:** discover the whole story, build the right product. 1. ed. O'Reilly Media, Inc., 2014.

SUTCLIFFE, Alistair; SAWYER, Pete. Requirements elicitation: Towards the unknown unknowns. **Requirements Engineering Conference (**RE), 2013 21st IEEE International. IEEE, 2013.

WAGNER, Jeremy L. *Web Performance in Action*. Manning, 2014.

WEB Fundamentals. **Google Developers**. Disponível em: <https://developers.google.com/web/fundamentals/>. Último acesso: 26 out. 2021.