**Engenharia – Conceitos, Modelos de desenvolvimentos, Desen. Ágil**

De acordo com Sommerville, "a Engenharia de Software **não está relacionada apenas com os processos técnicos** de desenvolvimento de software, **mas também com atividades de gerenciamento** de projeto de software e desenvolvimento de ferramentas, métodos e teorias que apoiem a produção de software.”.

O ciclo de vida de um software, entre outras características, está relacionado aos estágios de concepção, projeto, criação e implementação. São quatro atividades fundamentais do processo de software:

* Especificação de Software: são definidas as funcionalidades do software e restrições para sua operação.
* Projeto e Implementação de Software: o software que atenda à especificação deve ser produzido.
* Validação de Software: o software deve ser avaliado para garantir que ele faça o que o cliente deseja.
* Evolução do Software: o software evolui para atender às necessidades de mudança do cliente.

**De acordo com Sommerville, cada volta na espiral é dividida em quatro setores:**

Definição dos objetivos:

* determina objetivos específicos para essa fase do projeto
* identifica restrições ao processo e ao produto
* elabora um plano de projeto detalhado
* identifica os riscos do projeto

Avaliação e redução de riscos:

* é feita uma análise detalhada para cada risco identificado
* avalia alternativas, identifica e resolve os riscos

Desenvolvimento e validação:

* seleciona um modelo de desenvolvimento para o sistema (melhor abordagem de desenvolvimento, de acordo com os riscos identificados
* desenvolve e verifica o próximo nível do produto

Planejamento:

* revisa o projeto e planeja a próxima fase, ou seja, a próxima volta na espiral.

**De acordo com Pressman, cada volta na espiral é dividida em cinco setores:**

* Comunicação
* Planejamento: estimativa de custos, cronograma e análise de riscos
* Modelagem: análise e projeto
* Construção: codificação e testes
* Emprego: entrega e feedback

O **Manifesto Ágil** possui 4 valores e 12 princípios que fundamentam o desenvolvimento ágil de software. Os 4 valores são os seguintes:

* **Indivíduos e interações acima de processos e ferramentas**;
* Software em funcionamento acima de documentação abrangente;
* Colaboração com o cliente acima de negociação de contratos;
* Responder a mudanças acima de seguir um plano.

**Modelo “RAD”** - Desenvolvimento Rápido de Aplicações/ Requisitos bem compreendidos. É um modelo incremental. Quando usar: Ciclo de desenvolvimento extremamente curto.

* A aplicação não necessita de software auxiliares (Standalone)\* mais fechadas, não precisa interagir com outras aplicações.
* E possível fazer uso de classes pré-existentes
* A performance não é o mais importante
* O risco técnico reduzido
* A distribuição do produto no mercado é pequena
* O escopo do projeto é restrito
* O sistema pode ser Dividido em vários módulos
* O risco de mudança tecnológica é baixo
* Linguagem de desenvolvimento Rápido Chamados de linguagem de 4º geração, Interação com usuário – linguagem visual, gerar códigos arrastar e soltar.
* É um modelo de processo incremental que assume um ciclo de desenvolvimento curto e utiliza uma abordagem de construção com base em componentes.

Vantagens:

* Permite o desenvolvimento rápido e/ou a prototipagem de aplicações
* Reutiliza componentes
* Desenvolvimento em alto nível de abstração
* Redução na codificação manual (Wizards) \*automatizado clicando em next ate o final
* Envolvimento maior do usuário.

Desvantagens:

* Custo alto do conjunto de ferramentas
* Perda de precisão científica
* Pode levar ao retorno de praticas caóticas de desenvolvimento.
* Pode construir funções desnecessárias.

**TDD** - é uma técnica de desenvolvimento de software que se relaciona com o conceito de verificação e validação e se baseia em um ciclo curto de repetições: Primeiramente o desenvolvedor escreve um caso de teste automatizado que define uma melhoria desejada ou uma nova funcionalidade. Então, é produzido código que possa ser validado pelo teste para posteriormente o código ser refatorado para um código sob padrões aceitáveis. O Test-Driven Development (TDD) é um método ágil de desenvolvimento de software que se baseia na repetição de um ciclo de desenvolvimento curto, focado em testes unitários, em que os casos de teste que verificam uma nova funcionalidade são escritos antes mesmo da própria funcionalidade. Escreve-se o teste, encontre uma falha e refatore-o ciclicamente – conhecido como Red, Green e Refactor.

**Modelo em espiral** – Em todas as evoluções (volta na espiral) os riscos são considerados e documentados: "Como o software evolui à medida que o processo avança, o desenvolvedor e o cliente compreendem e a reagem melhor aos riscos em cada nível evolucionário" (Pressman). Segundo Pressman, os modelos Evolucionários (ou Evolutivos) são: Prototipação (Ajuda na descoberta de requisitos) e Espiral (voltada a Riscos). Modelo espiral é iterativo e evolucionário [PRESSMAN] e, não, incremental. Logo, não há necessariamente a entrega de incrementos funcionais ao final de cada fase. Falou de *Análise de Risco* **e** *Geração de Protótipos,*provavelmente teremos uma questão falando sobre Espiral de Boehm. Modelo espiral é iterativo e evolucionário [PRESSMAN] e, não, incremental. Logo, não há necessariamente a entrega de incrementos funcionais ao final de cada fase.

**A espiral é composta de**:

* Comunicação;
* Planejamento (estimativa, cronograma, análise de risco);
* Modelagem (análise projeto);
* Construção (código, teste);
* Entrega (entrega, feedback);

Éum processo evolucionário, ou seja, adequado para softwares que precisam passar por inúmeras evoluções na medida em que o desenvolvimento acontece.

Diferente do *Modelo Incremental*, que entrega partes prontas uma de cada vez, o *Modelo Espiral* é mais iterativo e tenta fazer sucessivos refinamentos. Outras novidades são os novos conceitos de ***Prototipagem* e *Gerenciamento de Riscos***. O *Modelo em Espiral* “*combina prevenção e tolerância a mudanças, assume que mudanças são um resultado de riscos de projeto e inclui atividades explícitas de gerenciamento de riscos para sua “redução*”, *uma abordagem realista do desenvolvimento de sistemas e softwares de grande porte… usando a prototipagem como mecanismo de redução de riscos*”.

A principal diferença entre o modelo em espiral e os outros modelos de processo de software é seu reconhecimento **explícito dos riscos**. Cada volta na espiral é dividida em quatro setores:

* **Definição de Objetivos:** Objetivos específicos para essa fase do projeto são definidos; restrições ao processo e ao produto são identificadas, e um plano de gerenciamento detalhado é elaborado.
* **Avaliação e redução de riscos:** Para cada um dos riscos identificados no projeto, é feita uma análise detalhada.
* **Desenvolvimento e validação:** Após a validação dos riscos, é selecionado um modelo de desenvolvimento para o sistema.
* **Planejamento:** O projeto é revisado, e uma decisão é tomada a respeito da continuidade do modelo com mais uma volta na espiral.

**Modelo em cascata** - o modelo em cascata, algumas vezes chamado ciclo de vida clássico, é um exemplo de processo dirigido a planos, pois se devem planejar todas as atividades (estágios) do processo antes de começar a trabalhar nelas. Em princípio, o estágio seguinte não deve ser iniciado até que o estágio anterior seja concluído, mas na **prática este processo não é um modelo linear simples**, envolvendo o feedback de um estágio a outro. Assim os documentos e artefatos produzidos em cada estágio podem ser modificados para refletirem as alterações em cada um deles. Este modelo é consistente com outros modelos de processo de engenharia, e a documentação é produzida em cada estágio do ciclo. Desta forma, o processo torna-se visível e os gerentes podem monitorar o progresso de acordo com o plano de desenvolvimento. Seu maior problema é a divisão inflexível do projeto em estágios distintos e, por isso, deve ser usado apenas quando os requisitos são bem compreendidos e pouco provavelmente venham a ser radicalmente alterados durante o desenvolvimento.

" Uma **variação na representação do modelo cascata é denominada modelo V**.... Descreve a **relação entre ações de garantia da qualidade** e as **ações associadas a comunicação**, **modelagem e atividades de construção iniciais.** .... Uma vez que o código tenha sido gerado, a equipe se desloca para cima, no ado direito do V, realizando basicamente uma série de testes (ações de garantia da qualidade) que validem cada um dos modelos criados à medida que a equipe se desloca para baixo, no lado esquerdo do V...."

Fases do modelo em Cascata:

* 1º - *Análise e definição de requisitos;*
* 2º - *Projeto de sistema e software;*
* **3º - *Implementação e teste de unidade;***
* **4º - *Integração e teste de sistema;***
* 5º - *Operação e manutenção;*

**Devops** é um termo criado para descrever um conjunto de práticas para integração entre as equipes de desenvolvimento de softwares, operações (infraestrutura ou sysadmin) e de apoio envolvidas (como controle de qualidade) e a adoção de processos automatizados para produção rápida e segura de aplicações e serviços. Propõe novos pensamentos sobre o trabalho para a valorização da diversidade de atividades e profissionais envolvidos e atitudes colaborativas. É um processo que torna possível o desenvolvimento ágil de aplicações em um modelo de gestão de infraestrutura definido sob regras rígidas e burocráticas.

A cultura Devops sustenta-se nos seguintes pilares:

* **Integração Contínua**: fácil transferência de conhecimento e experiências entre as áreas de Desenvolvimento, Operações e Apoio.
* **Implantação Contínua**: liberação rápida e continua de novas versões de software ou serviços.
* **Feedback contínuo:** feedbacks frequentes das equipes envolvidas em todas as fases do ciclo de vida do software ou serviço.

Características Devops:

* Colaboração e desenvolvimento de empatia entre integrantes das equipes com foco no projeto e não em interesses pessoais.
* Reuniões com a participação de profissionais de várias áreas necessárias para o projeto (desenvolvimento, operações, apoio).
* Implementação do gerenciamento de configuração para que mudanças realizadas manualmente nos servidores, sem conhecimento da gerência de configurações, sejam desfeitas.
* Estratégias para gestão de incidentes bem definidas, políticas de rollback, backup e ferramentas de monitoração proativas.
* Ambientes necessários para o trabalho da equipe de desenvolvimento providos de forma dinâmica e automatizada, sem necessidade de intervenção da equipe de operações.

Devops é um termo criado para descrever um conjunto de práticas para integração entre as equipes de desenvolvimento de softwares, operações (infraestrutura ou sysadmin) e de apoio envolvidas (como controle de qualidade) e a adoção de processos automatizados para produção rápida e segura de aplicações e serviços. O conceito propõe novos pensamentos sobre o trabalho para a valorização da diversidade de atividades e profissionais envolvidos e atitudes colaborativas. É um processo que torna possível o desenvolvimento ágil de aplicações em um modelo de gestão de infraestrutura definido sob regras rígidas e burocráticas.

Tradicionalmente Desenvolvimento e Operações são setores diferentes nas empresas e com motivações distintas. O setor de Desenvolvimento já evoluiu com adoção de metodologias ágeis e estão mais alinhadas ao negócio. O setor já consegue entregas rápidas e constantes para atender a expectativa dos clientes por novos recursos e assim valorizar o produto da empresa. A área de Operações, por sua vez, deseja o mínimo de alterações possíveis no ambiente de produção, pois podem gerar um novo ponto de instabilidade o que desvalorizará o produto da empresa. Assim há interesses contraditórios. Um setor quer evoluir e o outro garantir.

**A cultura DevOps sustenta-se nos pilares:**

* Integração Contínua: fácil transferência de conhecimento e experiências entre as áreas de Desenvolvimento, Operações e Apoio.
* Implantação Contínua: liberação rápida e continua de novas versões de software ou serviços.
* Feedback contínuo: feedbacks frequentes das equipes envolvidas em todas as fases do ciclo de vida do software ou serviço.

**Para alcançar esses objetivos a prática Devops recomenda ações como:**

* **Pessoas integradas:** Apoiar e prover pensamentos que integrem as pessoas, que façam com que partilhem suas histórias e se desenvolva a empatia entre elas para um trabalho conjunto eficaz e duradouro.
* **Foco no projeto:** Crie uma atmosfera livre de culpa, com o objetivo em comum: o projeto. Profissionais devem defender o projeto e não suas áreas de atuação. É preciso romper tradições e fazer com que as equipes tenham um comportamento colaborativo, construtivo e de respeito mútuo.
* **Reuniões conjuntas:**em vez de promover discussões isoladas com a equipe de desenvolvimento, operações ou apoio, sempre integre pelo menos um profissional de cada área nas discussões dos setores para que tenham entendimento dos objetivos a serem alcançados, recursos e demanda previstos, requisitos necessários, problemas já enfrentados e riscos envolvidos sob uma mesma ótica.
* **Negócio Just-in-Time:** Fornecimento de aplicações e serviços que promovam um desenvolvimento do negócio com qualidade e otimização do uso de recursos humano, tempo, tecnológicos e/ou financeiros.
* **Infraestrutura para negócio:** garantir continuamente a infraestrutura com foco no negócio. Implantar mecanismos que permitam a área de operações atenderem as expectativas do negócio e ainda sim manter sua confiabilidade.
* **Desenvolvimento Ágil:**o desenvolvimento do software deve seguir uma das metodologias ágeis para entregas rápidas e contínuas. ( SCRUM, XP, …)
* **Ambientes de Desenvolvimento, Homologação e Produção:** que haja pelo menos esses três ambientes e que sejam idênticos para evitar que uma versão de software seja testada em um ambiente e executada em produção em outro e assim surjam problemas não previstos.
* **Padronização nas configurações:** para garantia de que os ambientes sejam idênticos e contenham apenas mudanças homologadas, é preciso implementar um gerenciamento de configuração para que qualquer mudança inserida manualmente nos servidores e não  através de uma gerência de configurações seja automaticamente desfeita.
* **Provisionamento dinâmico dos ambientes:** os ambientes devem ser criados sempre que necessários em processos automatizados para garantia de que estejam sempre disponíveis. A equipe de desenvolvimento deve receber a infraestrutura necessária para seu trabalho sem necessidade de intervenção da equipe de operações. Ferramentas de automação deverão criar servidores, instalar serviços, configurá-los e testá-los. Novos servidores poderão ser criados temporariamente para ações específicas ou para escalonamento da solução.
* **Infraestrutura como um código:** as configurações e scripts de execução para instalação de serviços devem ser versionados no mesmo repositório e da mesma forma que o código da aplicação para que possam ser disponibilizados, auditados e evoluídos juntos.
* **Liberdade para Deploy:** a equipe de desenvolvimento deve ser autônoma para realização de deploy nos ambientes, até produção sem necessidade de processos burocráticos e interferência da área de operações.
* **Integração contínua:** Ferramentas devem orquestrar todo o processo envolvido na entrega de nova versão da aplicação que inclui a criação dos ambientes caso necessário, deploys dos códigos juntamente as configurações da infra, testes automatizados, possibilidade de reversão e auditoria.
* **Gestão de incidentes:** Para que a infraestrutura seja ágil é determinante que haja estratégias para gestão de incidentes bem definidas, políticas de roll back, backups e ferramentas de monitoração proativas.

**Desenvolvimento de software Adaptativos**

* É uma técnica para construção de software e sistemas complexos.
* Concentra-se na colaboração humana e na **auto-organização** das equipes.
* Define um ciclo de vida que incorpora três fases: **especulação**, **colaboração** e **aprendizagem**.
* A ênfase está na dinâmica das equipes **auto organizadas**, na colaboração interpessoal e na aprendizagem individual que levam as equipes de projeto de software a uma probabilidade maior de sucesso.

Jim Highsmith propôs o Desenvolvimento de Software Adaptativo (Adaptative Software Development - ASD) como uma técnica para construção de software e sistemas altamente complexos. Esse modelo se concentra na colaboração e auto-organização das equipes.

Na fase de **especulação** o projeto é iniciado e tem-se o planejamento de ciclos adaptáveis. Esse planejamento de ciclos adaptáveis usa as informações contidas no inicio do projeto como: a missão do cliente, restrições do projeto e os requisitos básicos.

A **colaboração** envolve confiança, críticas sem animosidade, auxílio, trabalho árduo, comunicação dos problemas ou preocupações de forma a conduzir ações efetivas, etc.

O **aprendizado** está no fato de que, com o andamento do projeto, o desenvolvedor passa a conhecer os desejos do cliente, adquirindo experiência e domínio do assunto e, consequentemente, da aplicação, além de conhecer antecipadamente os próximos passos a serem desenvolvidos. Os ciclos de revisões e testes devem ser curtos o suficiente para aprender apenas com pequenos erros, não oferecendo grandes riscos ao projeto.

**Kanban** auxilia que a equipe veja o progresso das atividades e permite que correções no processo atual da empresa sejam feitas, não definindo papéis muito menos ciclos de desenvolvimento.

Já dizia o ditado que uma imagem vale mais do que mil palavras. E a ciência comprova isso. A capacidade do cérebro de processar informações visuais é muito maior do que a de processar informações textuais. Mas muito maior mesmo!

Usando o Kanban, é possível ver todo o trabalho e entender com mais facilidade quais tarefas precisam ser realizadas e quais já foram cumpridas. Assim, a ferramenta permite que você observe o fluxo e consiga identificar gargalos e filas.

Quando você olha para seu processo graficamente, fica mais fácil organizar e limitar a quantidade de tarefas em processo ou inacabadas e com isso priorizar atividades. Outra grande vantagem é que ele facilita a troca de informações, contribuindo para uma cultura de colaboração dentro da empresa.

Você vai notar que, conforme for usando o sistema, conseguirá detectar problemas escondidos, atrasos e falhas em sua gestão. Assim, essa é uma ferramenta que também te ajuda a encontrar soluções mais eficientes e melhorar seus processos.

A entrega de um incremento de software representa um marco importante para qualquer projeto de software. **Um conjunto de princípios** essenciais **deve ser seguido** enquanto a equipe se prepara**para a entrega de um incremento**:

* Princípio 1: As expectativas do cliente para o software devem ser gerenciadas.
* Princípio 2: Um pacote de entrega completo deve ser montado e testado.
* Princípio 3:  É preciso estabelecer uma estrutura de suporte antes da entrega do software.
* **Princípio 4: Material instrucional adequado deve ser fornecido aos usuários.**
* Princípio 5: Software com bugs deve ser primeiramente corrigido e, depois, entregue.

**Segundo PMBoK 6:** As definições de ciclo de vida incremental e iterativo são bem diferentes na 6ª edição:

* **Ciclo de vida incremental**/ Incremental Life Cycle. Um ciclo de vida do projeto adaptativo no qual a entrega é produzida através de uma série de iterações que sucessivamente adicionam funcionalidade dentro de um prazo predeterminado. A entrega inclui a capacidade necessária e suficiente para ser considerada completa somente após a iteração final.
* **Ciclo de vida iterativo** / Iterative Life Cycle. **Ciclo de vida do projeto em que o escopo do projeto é geralmente determinado no início do ciclo de vida do mesmo, mas as estimativas de tempo e custos são rotineiramente modificadas à proporção que a compreensão do produto pela equipe do projeto aumenta.**Iterações desenvolvem o produto através de uma série de ciclos repetidos, enquanto os incrementos sucessivamente acrescentam à funcionalidade do produto.
* **Ciclo de vida preditivo** / Predictive Life Cycle. Uma forma de ciclo de vida do projeto na qual o escopo, tempo e custo do projeto são determinados nas fases iniciais do ciclo de vida.

Fonte: PMBOK 6ª edição, p. 702.

**Valores da Metodologia Ágil**

• Indivíduos e interações são mais importantes que processos e ferramentas.

• Software funcionando é mais importante do que documentação completa e detalhada.

• Colaboração com o cliente é mais importante do que negociação de contratos.

• Adaptação a mudanças é mais importante do que seguir o plano inicial.

**Princípios do Manifesto Ágil**

Nós seguimos os seguintes princípios:

* Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.
* Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.
* Entregar software funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.
* Pessoas relacionadas à negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.
* Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.
* 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.
* Software funcional é a medida primária de progresso.
* Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.
* Contínua atenção à excelência técnica e bom design, aumenta a agilidade.
* Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.
* As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis.
* Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

Walkthrough são tipos de revisão.

* Procedimento similar ao procedimento para condução de uma inspeção.
* A diferença fundamental está na maneira como a sessão de revisão é conduzida.
* Em vez de simplesmente ler o programa ou checar os erros por meio de um checklist, os participantes simulam sua execução.

**Lean** é um modelo gerencial aplicado através de princípios e técnicas operacionais, tendo como objetivo a redução do desperdício, a melhoria da qualidade e a maximização do valor entregue ao cliente. Em sua essência, é uma filosofia orientada à eficiência e eficácia de processos, centrada em criar mais valor com menos trabalho.

Basicamente os princípios são:

### 1) Eliminar o desperdício - São considerados como desperdícios:

* **Trabalho parcialmente feito** – O famoso “está pronto, só falta testar”. É fundamental que as equipes criem uma boa definição de “pronto/done” para evitar retrabalhos em itens teoricamente finalizados.
* **Processos extras** – Documentação “pesada” exigida que não agrega valor ao software final. Veja minhas considerações sobre estes tipos de processos extras [neste artigo](https://www.profissionaisti.com.br/2013/10/reflexoes-sobre-o-manifesto-agil-parte-2-software-funcional/).
* **Funcionalidades extras**– O famoso “Gold-Platting”, ou seja, desenvolver funcionalidades não requeridas pelo seu cliente/usuário final visando agradá-lo. Você pode me perguntar: “Mas Vitor, não temos que encantar nosso cliente e colaborar conforme você mesmo escreveu [neste artigo](https://www.profissionaisti.com.br/2013/10/reflexoes-sobre-o-manifesto-agil-parte-3-colaboracao-com-o-cliente/)?”. Minha resposta: “Encante seu cliente, entregando **VALOR** e **QUALIDADE** na construção do software baseado nos requisitos que ele pediu”.
* **Alternação de tarefas ou multi-tarefas** – O famoso “atue 20% do tempo neste assunto,  atue 37,5% do tempo neste outro assunto, atue 22,4% do tempo neste outro assunto e o restante do tempo neste assunto que é mais prioritário”. É cientificamente comprovado que até 40% do esforço é desperdiçado em alternar tarefas, pois o cérebro possui um “delay” para se desligar de uma tarefa e se concentrar em outra. Quando tentamos fazer tudo de uma vez, na verdade não estamos dando foco em nada.
* **Espera** – A equipe desenvolve, faz os testes unitários, realiza a garantia da qualidade e quando passa o software para homologação, o cliente não tem tempo para testar. Esta espera é muito custosa para um projeto de software, pois os recursos de desenvolvimento não podem ser liberados enquanto não finalizar o projeto, porém ficam ociosos.
* **Esforços de comunicação**– Equipes grandes ou geograficamente distribuídas requerem uma boa gestão de comunicação e uma boa [gestão de stakeholders](https://www.profissionaisti.com.br/2013/10/gerenciamento-dos-stakeholders-uma-arte/). A falha nessa gestão pode comprometer o projeto e o tempo gasto para colocar o projeto de volta aos trilhos pode ser considerado como desperdício.
* **Defeitos**– Quer pior coisa do que implementar um software cheio de bugs para o seu cliente? A credibilidade do time vai lá para baixo, além de gerar um custo de qualidade para a correção destes bugs e gastar tempo que poderia ser utilizado em outros projetos.

### 2) Fortalecer o time - Criar um ambiente onde a equipe trabalhe de forma [auto-organizada e auto-dirigida](https://www.profissionaisti.com.br/2013/11/auto-gerenciamento-versus-auto-organizacao-qual-o-perfil-de-uma-equipe-agil/), evitando [micro-gerenciamento](https://www.profissionaisti.com.br/2013/09/scrum-10-situacoes-de-quando-ele-podera-e-certamente-ira-fracassar/)

### 3) Entregas rápidas - Maximizar o ROI (Return Of Investiment) do projeto, entregando software de valor de forma rápida e contínua.

### 4) Otimizar o todo - Entender que o software concluído é muito mais que a soma das partes entregues e verificar como ele está alinhado com os objetivos da empresa.

### 5) Construir qualidade - Garantir qualidade no desenvolvimento do software utilizando técnicas como:

* Teste unitário através de TDD : Escrever o teste / teste falho / escrever o código para passar o teste
* Refatoração: Melhorar e refinar o código escrito, evitando manter legados de débitos técnicos e fazendo com o que o software continue se comportando da mesma maneira.
* Integração contínua: Compilação automática do repositório de código, verificando se o código inserido não prejudicou demais funcionalidades do software.

### 6) Adiar decisões - Deixar as decisões e comprometimentos para o último momento responsável, permitindo coletar informações e ter experiências para fortalecer a tomada de decisão.

### 7) Amplificar conhecimento - Priorizar a comunicação e o feedback contínuos entre equipes e usuários durante o processo de desenvolvimento de software.

O que vejo hoje como grande falha, na maior parte dos projetos de desenvolvimento de software que vivenciei, em todos os pontos, é a questão do desperdício e da falta de qualidade. Bom, mas essa é a minha experiência! E a de vocês ? Compartilhem com os leitores do PTI!