**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.

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.

**XP – Extreming Programming** - em **XP**, um cliente ou usuário é parte do time de XP e é responsável na tomada de decisões sobre requisitos. XP também encoraja todos da equipe a usar os cartões **CRC (classe-responsabilidade-desenvolvedor)** que serve como um mecanismo bastante eficaz para pensar sobre o software em um contexto **OO** (orientado a objetos). Esses cartões permitem identificarmos e organizarmos as classes OO para o incremente sendo desenvolvido. Os cartões CRC são o único artefato de projeto produzido como parte do projeto XP. Fazem parte das principais práticas do XP:

* Planejamento Incremental
* Pequenas Releases - entregas pequenas
* Projeto Simples
* Desenvolvimento Test-First - primeiro os testes
* Refatoração
* Programação em pares
* Propriedade coletiva
* Integração contínua
* **Ritmo sustentável - 40h semanais**
* Uso de metáforas
* Cliente on-site - cliente sempre disponível
* Reuniões em pé
* Jogo do planejamento
* Time coeso

Planejamento incremental e Histórias do Usuário são práticas XP.

* **Planejamento Incremental (Planning Game):** requisitos são registrados como histórias dos usuários e priorizados para serem incluídos em uma determinada iteração.
* **Histórias de Usuário:**descrevem requisitos de forma ágil. São textuais e escritas em cartões. Segue o padrão: "Como um aluno, quero registrar meus livros para que..."
* **Práticas XP:** Metáfora, **Histórias de Usuário**, Projeto Simples, Refatoração, Programação em Pares, Propriedade coletiva do código, Padrão de codificação, Ritmo sustentável, Cliente sempre presente, Reuniões em pé, TDD, Integração contínua, **Planejamento incremental** e Time coeso.

**Plano de Iteração ou Iteration Planning Game:** Cada iteração é de 1 a 3 semanas. Histórias de usuários são escolhidas para essa iteração pelo cliente no plano de liberação, na ordem de maior valor para o cliente primeiro. Testes de aceitação com falha a serem corrigidos também são selecionados.

As histórias do usuário e os testes com falha são divididos nas tarefas de programação que os darão suporte. Enquanto as histórias do usuário estão no idioma do cliente, as tarefas estão no idioma do desenvolvedor (CRC). Esses cartões de tarefas serão o plano detalhado da iteração.

**Os desenvolvedores se inscrevem para executar as tarefas e depois estimam quanto tempo suas próprias tarefas levarão para serem concluídas.** É importante que o desenvolvedor que aceita uma tarefa também seja quem calcula quanto tempo levará para concluir. As pessoas não são intercambiáveis ​​e a pessoa que fará a tarefa deve estimar quanto tempo levará.

Cada tarefa deve ser estimada em 1, 2 ou 3 (adicione 1/2, se necessário) dias de programação ideais. Os dias ideais de programação são quanto tempo você levaria para concluir a tarefa se não houvesse distrações. Tarefas com duração inferior a 1 dia podem ser agrupadas. Tarefas com duração superior a 3 dias devem ser divididas mais adiante.

Fonte: http://www.extremeprogramming.org/rules/iterationplanning.html

Cuidado para não confundir **iteration planning** com **planning poker**. Ele pode ser usado para estimar os cartões, mas não é citado explicitamente nas regras. É apenas uma das práticas citadas.

**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 se 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 da 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 todos 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 deve-se 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.

**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.

**Desenvolvimento de software Adaptativos**

* É uma técnica para construção de software e sistemas complexos.
* Se concentra 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.

**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 sufciente para ser considerada completa somente após a iteração fnal.

**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 modifcadas 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.