PEF – 5743 – Computação Gráfica Aplicada à Engenharia de Estruturas

Prof. Dr. Rodrigo Provasi

e-mail: provasi@usp.br

Sala 09 – LEM – Prédio de Engenharia Civil

Projeto de *Software – Parte I*

- Ciclo de vida de uma aplicação
- Engenharia de Requisitos
- Métodos para criação de *software*

Introdução

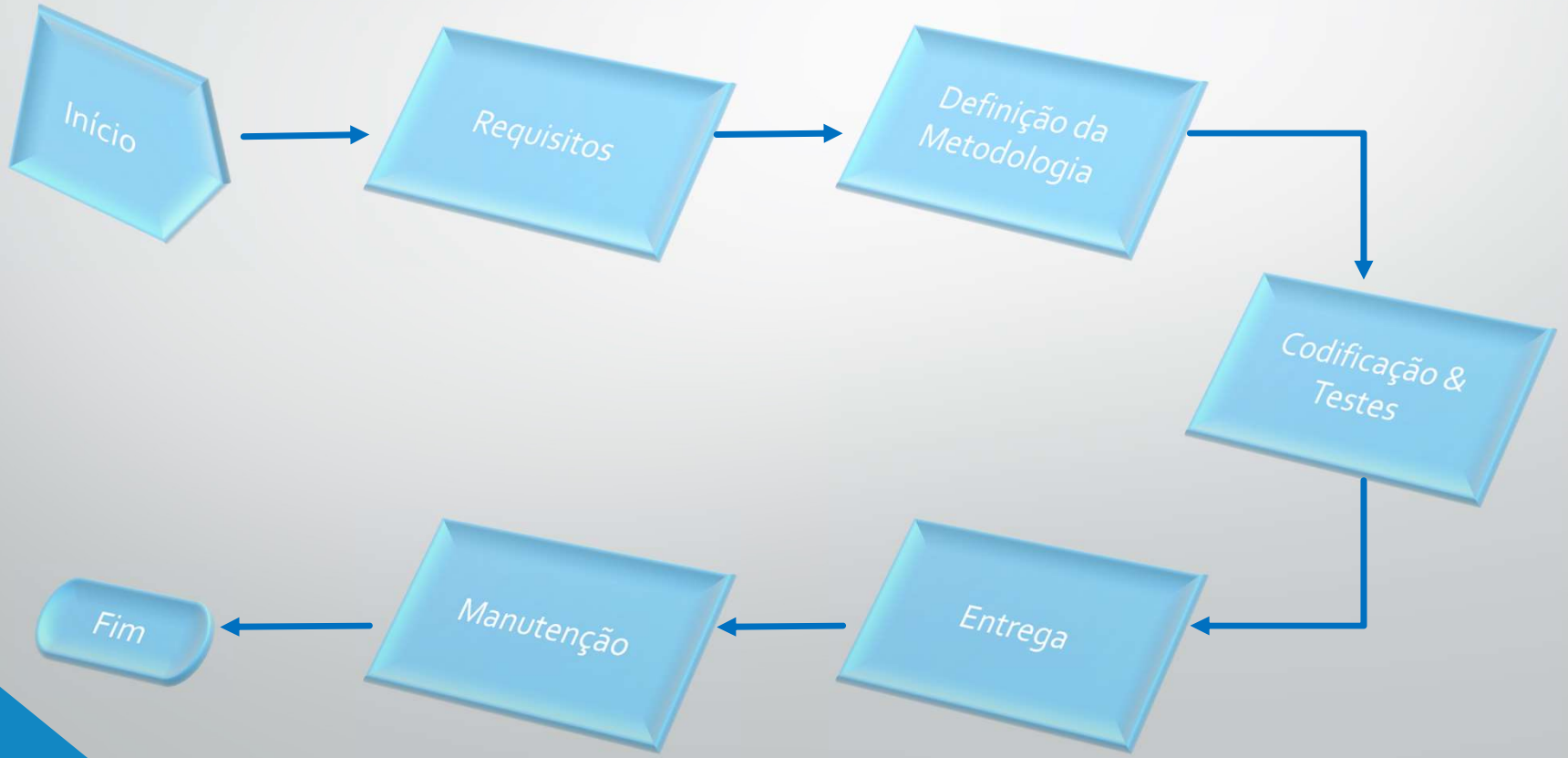
- Eventualidades que ocorrem durante o desenvolvimento de um *software*
  - Detalhes não previstos no projeto inicial
  - Mudança por parte dos clientes
  - Problemas de codificação
- Bom projeto → Redução do impacto dessas eventualidades e da quantidade de erros → Evitar redundâncias e retrabalho.

Introdução – O que é *Software*?

- *Software* é desenvolvido; não é manufaturado no sentido clássico.
- *Software* não "desgasta".
- Embora a indústria move-se em direção à construção baseada em componentes, a maioria dos programas continua sendo customizada

- Por IEEE Standards:
  - 1) A aplicação de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, operação e manutenção de *software*; isto é, a aplicação da engenharia de *software*.
  - 2) O estudo das técnicas como em 1)

Ciclo de vida de uma aplicação



Engenharia de Requisitos

- Abrange o levantamento de todas as funcionalidades desejadas no *software* pelo cliente.
- Leva a:
  - Aumento da qualidade
  - Definição de ferramentas, linguagens, documentação adequadas.
  - Maior interação com o cliente (até a entrega do produto).

Engenharia de Requisitos

- Pode ser dividida em:
  - **Requisitos do Usuário:** são declarações, em linguagem natural e também em diagramas, listando as funções desejadas pelo usuário.
  - **Requisitos de Sistema:** estabelecem detalhadamente as funções e restrições do sistema.
  - **Especificação do Projeto de *Software*:** é uma descrição abstrata do projeto de *software*, servindo como base para um projeto e especificação mais detalhados.

Engenharia de Requisitos

- Outra possível classificação:
  - **Requisitos Funcionais:** são declarações de quais funções o sistema deve executar, de como o sistema deve reagir a entradas específicas e de como deve se comportar em determinadas situações.
  - **Requisitos Não funcionais:** são restrições sobre os serviços ou as funções oferecidas pelo sistema. São ligados a características internas, não externadas pelo *software*.
  - **Requisitos de Domínio:** são requisitos que se originam do domínio da aplicação (como por exemplo o sistema operacional) e refletem características desse domínio. Podem ser requisitos funcionais ou não funcionais.

Requisitos de Usuário

- Descrevem os requisitos do projeto (tanto funcionais quanto não funcionais)
- Descrição de forma natural (texto que uma pessoa consiga compreender).
- Problemas mais comuns:
  - Falta de clareza
  - Os requisitos, objetivos e informações podem não estar claramente definidos

Requisitos de Sistema

- São descrições mais detalhadas dos requisitos de usuário, que são utilizados pelos engenheiros de software como ponto de partida para o projeto de sistema.
- Não devem ter detalhes de codificação, e sim apenas o que o programa deve fazer.
- Pode conter detalhes de codificação se há necessidade de interoperabilidade entre o sistema a ser desenvolvido com os já desenvolvidos; a definição de uma arquitetura inicial que colabora e facilita para estruturar a especificação dos requisitos; ou ainda o uso de um projeto específico.

Requisitos de Sistema

- São escritos em linguagem natural → Cuidado com ambiguidades!
- Possíveis notações:
  - Linguagem natural estruturada
  - Linguagem de descrição de projeto
  - Notações gráficas
  - Especificações matemáticas

Especificação do Projeto de Software

- Características:
  - Especificar somente o comportamento externo do sistema.
  - Especificar as restrições à codificação.
  - Ser fácil de ser modificado.
  - Servir como ferramenta de referência para os responsáveis pela manutenção do sistema.
  - Registrar a estratégia sobre o ciclo de vida do sistema.
  - Caracterizar respostas aceitáveis para eventos indesejáveis.

Requisitos Funcionais, Não Funcionais e de Domínio

- Quanto a esta classificação, tem-se que os requisitos funcionais são aqueles que descrevem as funcionalidades e características que o sistema deve exibir, ou seja, as características que o cliente gostaria de observar quando da utilização do software.
- Os requisitos não funcionais são aqueles que estão relacionados com as propriedades do sistema, principalmente em relação a características como confiabilidade e segurança, além de estarem ligados com restrições impostas ao sistema, como por exemplo, tipos e limitações na entrada de dados.
- Já os requisitos de domínio estão ligados ao domínio da aplicação, como por exemplo, uma limitação ou o uso específico de uma interface de entrada de dados ou um padrão de banco de dados a ser usado. Pode-se pensar, por exemplo, que o cliente já possui o banco de dados e as informações a serem utilizadas são provenientes desse; assim, a interface de comunicação é uma restrição do domínio do sistema.

Metodologias de Projeto

- A escolha da metodologia a ser empregada depende:
  - do projeto a ser desenvolvido
  - da duração do projeto
  - do público alvo
  - das ferramentas disponíveis
  - do pessoal
  - do conhecimento da equipe
  - das necessidades do cliente
  - etc.

Modelos Tradicionais

- Cascata
- Incremental
- *RAD*
- Prototipagem
- Espiral (ou Evolutivo)
- Baseado em componentes
- Métodos Formais
- *Unified Process (UI)*

Metodologias Ágeis

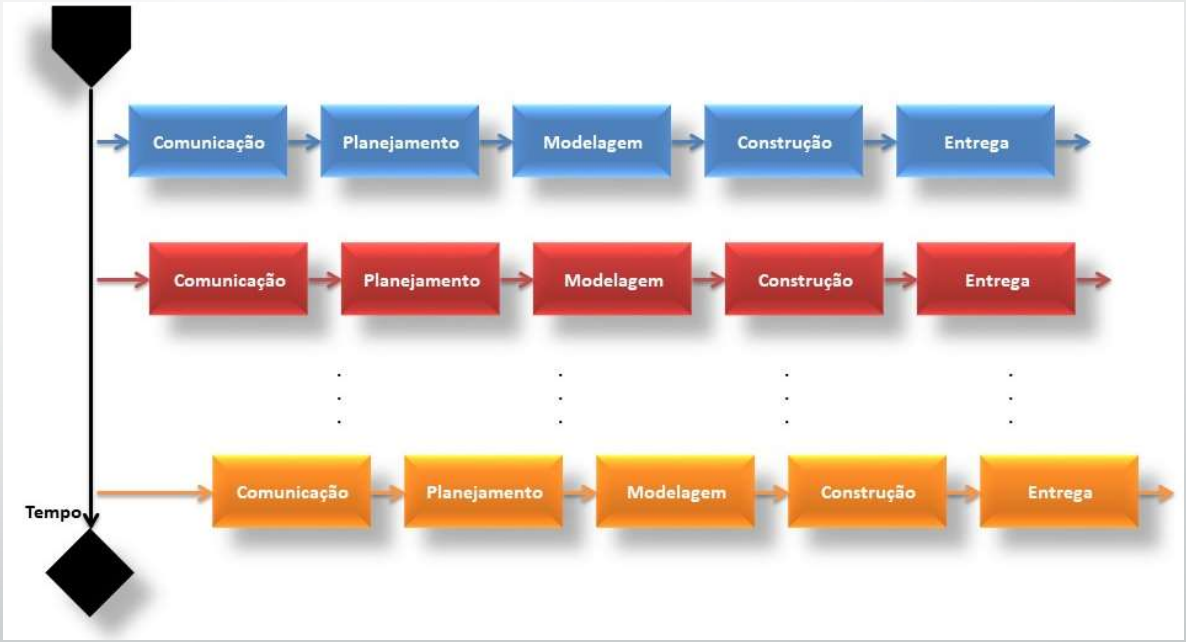
- *eXtreme Programming (XP)*
- Desenvolvimento Adaptativo
- Desenvolvimento Dinâmico
- *Scrum*

Modelo em Cascata

- Modelo Clássico
- Etapas:
  - **Comunicação:** etapa na qual o projeto é iniciado e os requisitos de projeto são levantados e reunidos.
  - **Planejamento:** etapa na qual há uma estimativa do projeto (tempo, custo, entre outros) e o agendamento das atividades.
  - **Modelagem:** etapa na qual a análise e o design do software são feitos.
  - **Construção:** etapa na qual se dá a codificação e os testes.
  - **Entrega:** etapa na qual há entrega do software, suporte e o feedback do usuário.

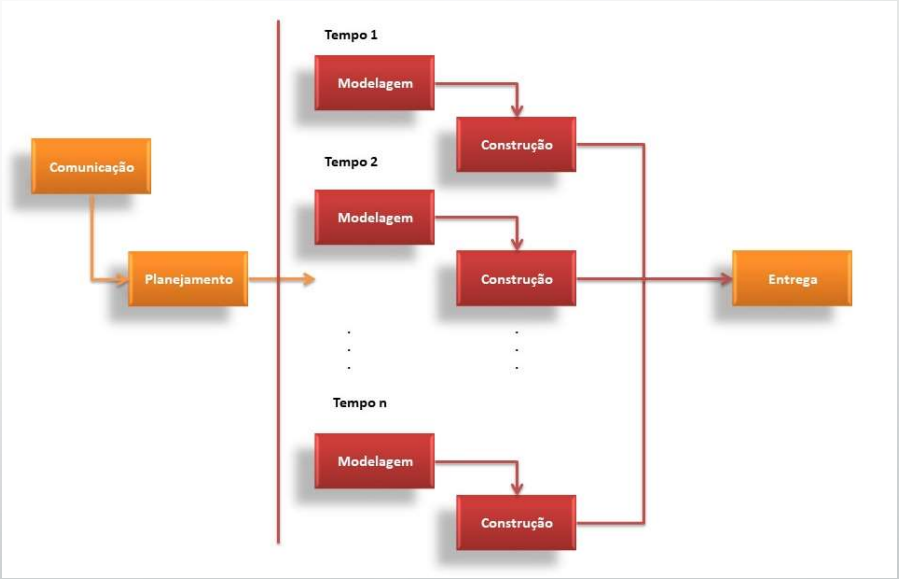
Modelo Incremental

- O modelo incremental combina as características lineares do modelo em cascata com divisões do desenvolvimento, separando o projeto e construção em várias iterações.



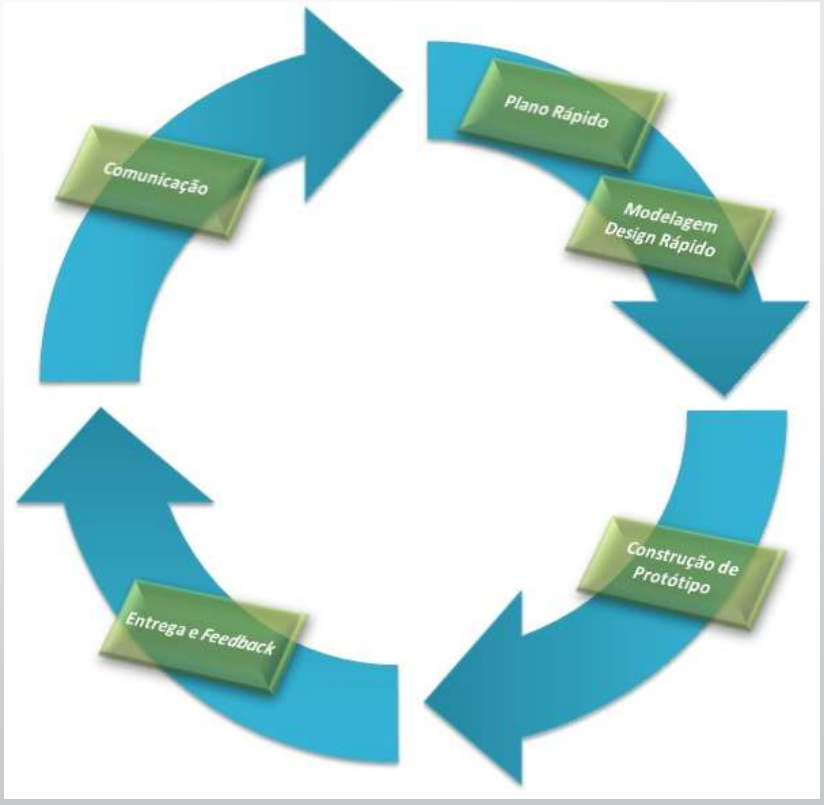
Modelo *RAD*

- O modelo RAD (acrônimo para *Rapid Application Development*) é um modelo de desenvolvimento incremental que busca desenvolvimento em um curto prazo.
- É uma implementação do modelo em forma de cascata no qual existem diversos times trabalhando paralelamente na modelagem e na construção



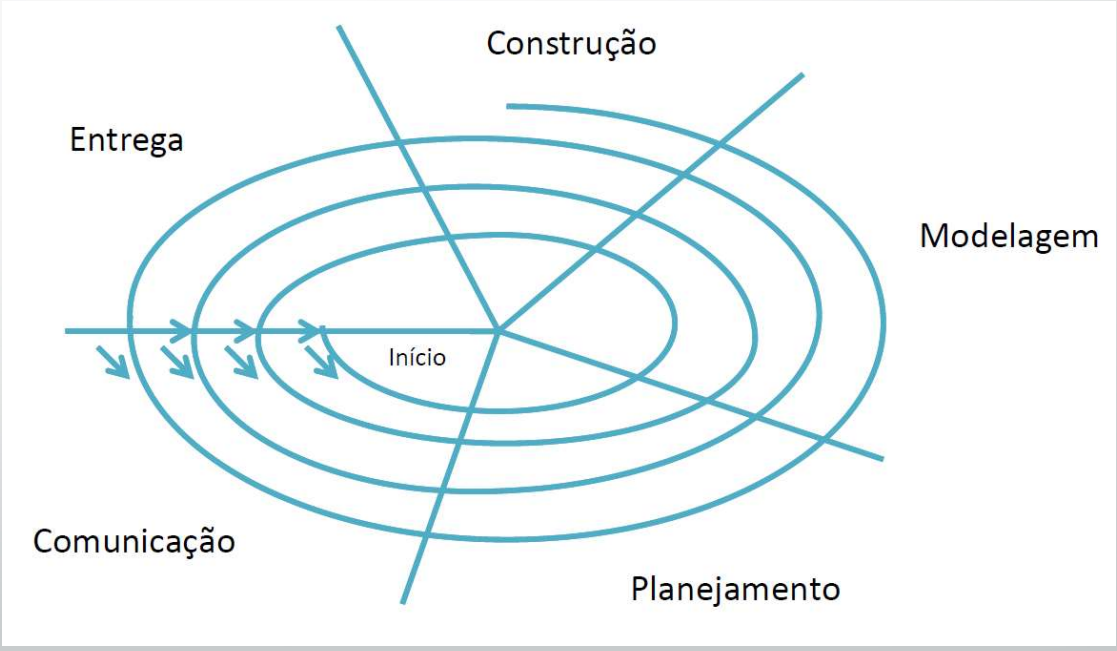
Prototipagem

- A prototipagem é um processo no qual o software é confeccionado rapidamente, com algumas das funcionalidades desejadas, para definir os requisitos de projeto que o usuário não conseguiu definir com precisão.
- São dados alguns requisitos gerais, é construído um protótipo e assim, através do feedback do cliente, o software pode ser construído.



Modelo Evolutivo ou Espiral

- O modelo em espiral ou evolutivo é um processo iterativo no qual nem todos os requisitos são conhecidos a priori.
- Em certos tipos de problema, os requisitos surgem conforme as iterações do software são apresentadas. Nesses casos, o modelo espiral é o mais indicado.



Desenvolvimento Baseado em Componentes

- O desenvolvimento baseado em componentes, que também segue uma filosofia de desenvolvimento iterativo, prega a reusabilidade de componentes, buscando, durante o projeto, utilizar-se de componentes prontos (chamados em inglês *off the shell*) para tornar o desenvolvimento mais rápido.
- Assim, um software é projetado de tal forma a utilizar-se da maior quantidade de componentes prontos possíveis, ao invés de construí-los.
- Caso seja necessária a confecção de um componente, esse é feito de tal maneira que seja reutilizável.

Métodos Formais

- Os métodos formais consistem em um grupo de atividades que levam a uma especificação do *software* que utiliza notações matemáticas.
- A grande vantagem desse método é que este fornece um mecanismo de eliminar a maioria dos problemas que nos outros métodos são de difícil solução → Elimina ambiguidades
- Em contrapartida, os métodos formais consomem muito tempo e recursos em um projeto, além de demandar uma equipe extremamente bem treinada.
- Difícil comunicação com o cliente (extremamente técnico).

Unified Process (UP)

- O processo unificado (ou no inglês *Unified Process*, conhecido pela sigla UP) procura conciliar as características dos processos convencionais de desenvolvimento de software e boa parte dos princípios do desenvolvimento ágil.
- O UP começa com a comunicação e planejamento de atividades. Nessa primeira etapa, há uma interação entre clientes e usuários finais, os requisitos do *software* são identificados e uma arquitetura rudimentar é definida.

Metodologias Ágeis

- Problemas nos modelos:
  - É difícil predizer quais requisitos de *software* irão permanecer inalterados e quais sofrerão mudanças, assim como é difícil predizer como as prioridades do cliente mudarão durante o projeto.
  - Para muitos tipos de *software*, design e construção são intercalados, sendo que devem ser feitos concomitantemente, permitindo que o *design* seja verificado conforme é criado. O problema encontrado é a dificuldade em predizer a quantidade de design necessária antes da construção.
  - Análise, *design*, construção e teste não são tão previsíveis quanto se gostaria.

Metodologias Ágeis

Características dos grupos e indivíduos nas metodologias ágeis:

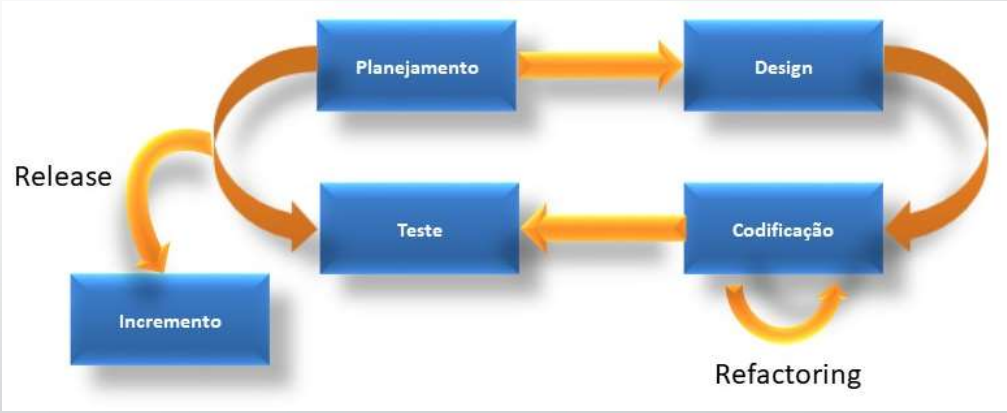
- **Competência:** engloba talento, habilidades relacionadas com software e conhecimentos gerais do processo escolhido pelo grupo. Habilidade e conhecimento do processo podem ser ensinados pelo líder do time.
- **Foco comum:** embora os membros do grupo executem tarefas diferentes e possuam habilidades distintas, todos devem focar no mesmo objetivo: entregar um incremento funcional do software para o cliente no prazo estabelecido. Para atingir esse objetivo, o time deve também focar em adaptações continuas que façam com que o processo se adapte às necessidades.
- **Colaboração:** os integrantes do time devem colaborar – e, consequentemente, comunicar-se – para que o objetivo seja concluído; essa colaboração gera informações importantes para o time (como por exemplo, o status atual do projeto, andamento de tarefas), bem como para o cliente (tanto o incremento quanto a documentação associada ao incremento).

Metodologias Ágeis

- **Habilidade de tomar decisões:** todo bom time de desenvolvimento de software deve ter a liberdade de tomar suas próprias decisões, tanto em aspectos técnicos quanto de projeto.
- **Habilidade de resolver problemas nebulosos:** os gerentes de *software* devem reconhecer que o time de desenvolvimento ágil deve lidar continuamente com ambiguidades e que nem sempre as soluções são alcançadas facilmente. O time tem de saber lidar com possíveis erros e aprender com eles.
- **Respeito e confiança mútua:** os integrantes do time devem possuir respeito e confiança mútua para perseverar juntos, ou seja, uma unidade coesa a fim de alcançar um objetivo comum.
- **Auto-organização:** o time deve se organizar para que o trabalho seja feito, para encontrar a melhor acomodação no ambiente de trabalho e deve também organizar a sua agenda para a melhor entrega dos incrementos do *software*. Alguns benefícios trazidos pela auto-organização do time são uma melhoria na colaboração e um aumento do moral do time.

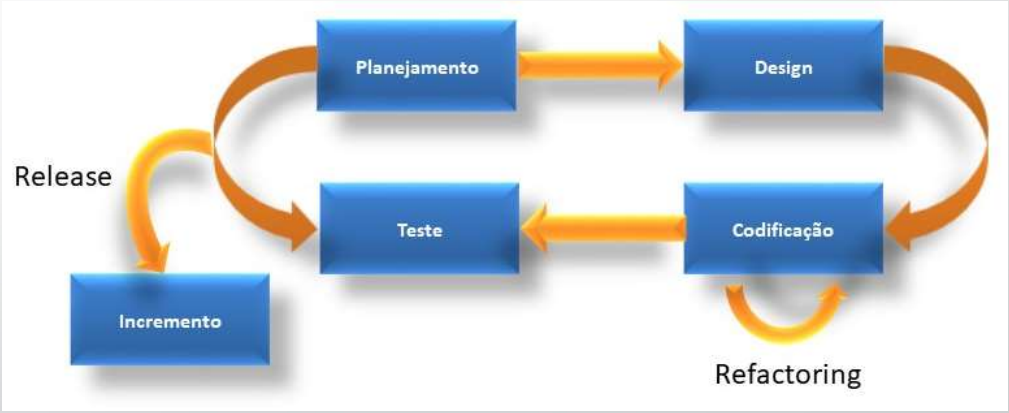
eXtreme Programming (XP)

- ***Planejamento:*** começa com a criação de um conjunto de histórias (conhecidas como *user stories*) que são colocadas em cartões numerados. O cliente atribui um valor para a história baseada no projeto (essa escala é previamente definida e varia de acordo com o projeto).



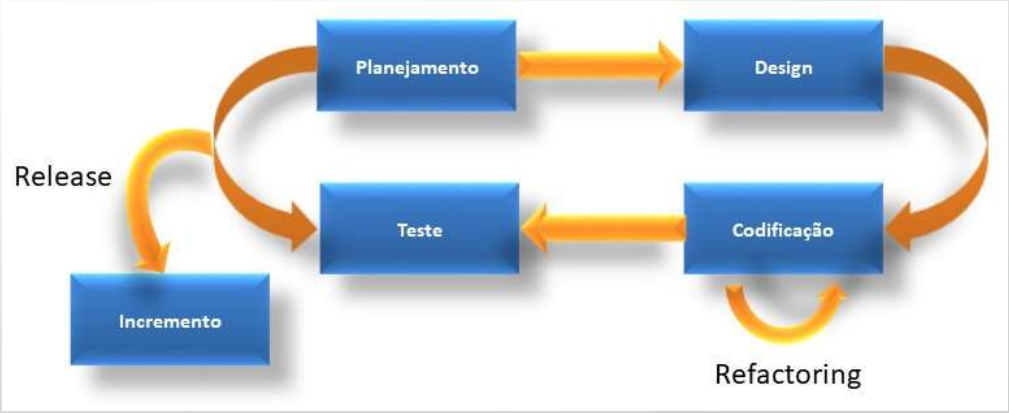
eXtreme Programming (XP)

- ***Design:*** a XP segue o princípio de manter o design simples, dando suporte à implementação das histórias. Para auxiliar o design no contexto de orientação a objetos é feito o uso de cartões CRC (do inglês *Card-Responsability Collaborator*).



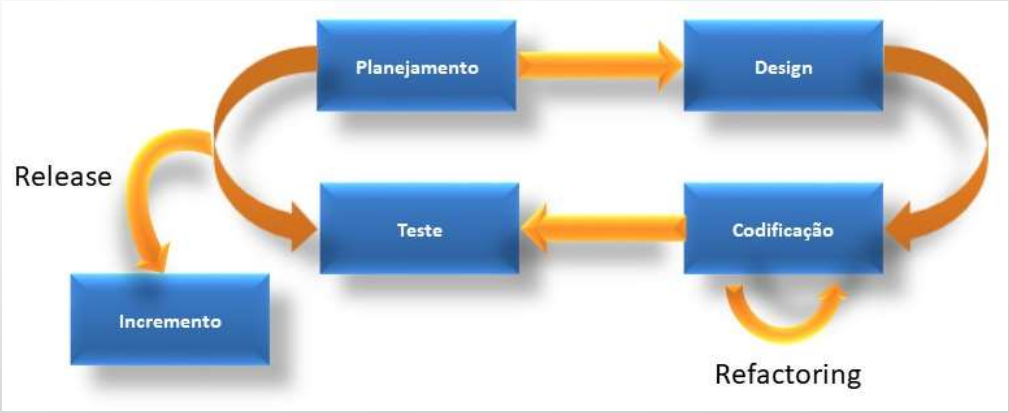
eXtreme Programming (XP)

- ***Codificação:*** a XP recomenda que após o desenvolvimento das histórias e do *design* preliminar, uma série de testes seja feita, de tal forma que durante a codificação seja possível focar no que deve ser implementado.



eXtreme Programming (XP)

- ***Teste:*** o teste na XP é iniciado durante a codificação. Nessa etapa são executados os testes de integração dos códigos gerados pelo time, devidamente integrados. Uma vez aprovado, o incremento é então liberado para o usuário.



Desenvolvimento Adaptativo de software

- O desenvolvimento adaptativo de software (conhecido pela sigla em inglês ASD – *Adaptative Software Development*) foca na comunicação, colaboração e auto-organização das pessoas envolvidas no desenvolvimento.
- Dividida em 3 estágios:
  - Especulação, o projeto é iniciado e o planejamento é feito (projeto ou incremento).
  - Colaboração é a etapa mais importante do processo, onde há a grande interação e colaboração entre os membros do time.
  - Aprendizado é a etapa na qual os membros do time avaliam o trabalho e procuram abalizar os erros e acertos nessa iteração, aprimorando assim o seu trabalho e o projeto final.

Método de Desenvolvimento Dinâmico de Sistemas

- O método de desenvolvimento dinâmico de sistemas (cuja sigla em inglês é DSDM - *Dynamic Systems Development Method*) é um método ágil similar ao método RAD que segue o princípio de que 80 por cento da aplicação devem ser entregues em 20 por cento do tempo (comparado com o tempo de entrega da tarefa concluída), sendo que ajustes e detalhes são acertados em iterações posteriores.

Scrum

- Princípios:
  - O uso de times pequenos organizados de tal forma a maximizar a comunicação e o compartilhamento de informações.
  - O processo deve adaptar-se tanto a mudanças técnicas quanto de negócio.
  - O processo deve produzir incrementos frequentemente.
  - O trabalho de desenvolvimento e as pessoas que nele trabalham devem estar dividas, mantendo a organização e clareza do projeto.
  - Testes e documentação são produzidos constantemente durante a execução do projeto.

Scrum

- Inicia-se o processo determinando as características que o software deve ter baseado na lista de requerimentos →Backlog
- Um Backlog é dividido em unidades menores de trabalho → Sprint
- Sprint é uma unidade de trabalho definida em uma janela de tempo, usualmente de 30 dias.
- Durante o Sprint reuniões diárias com duração de quinze minutos em média são realizadas.