



Faculdade
IMPACTA
TECNOLOGIA

Processo de Software

Pós-Graduação em Engenharia de Software





Disciplina Processo de Software 24h

04.11 a 09.12

Segundas-feiras





Processo de Software

Revisão



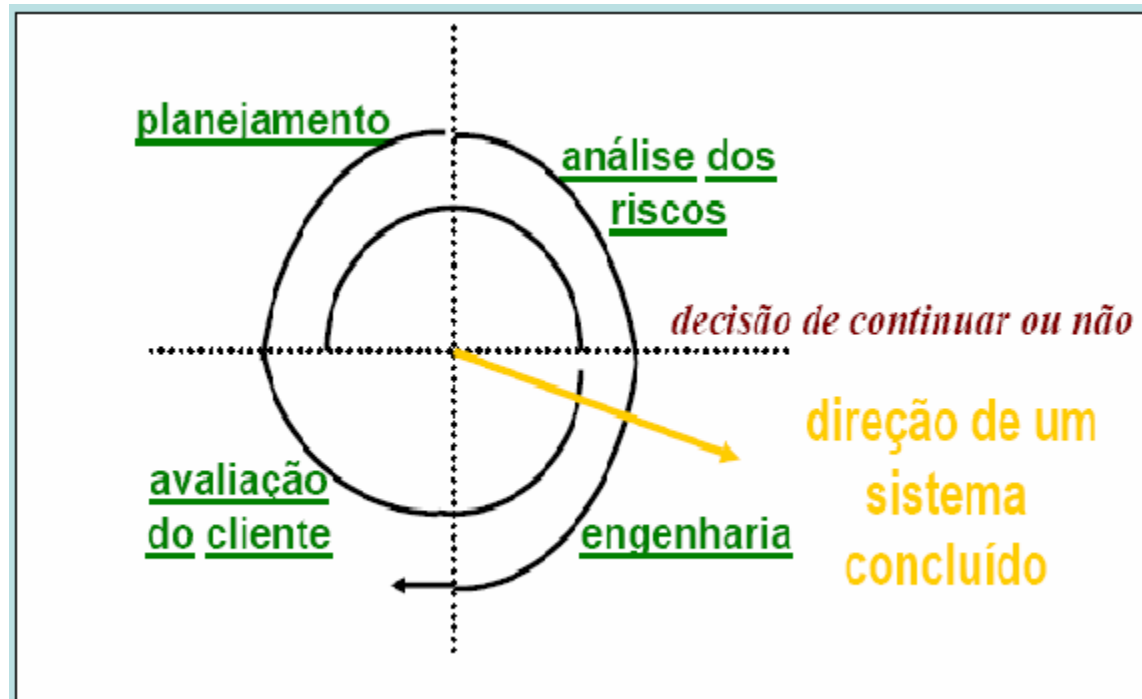
Modelos de Processo

- ✓ Codifica-Remenda;
- ✓ Clássico ou CASCATA / Waterfall;
- ✓ V-Model
- ✓ Sashimi;
- ✓ Espiral;
- ✓ Prototipagem Evolutiva;
- ✓ Entrega por Estágios;
- ✓ Entrega Evolutiva
- ✓ RAD (Rapid Application Development)
- ✓ Orientado a Reuso
- ✓ RUP* (Rational Unified Process)





Espiral





Espiral

- ✓ É desenvolvido em uma série de iterações;
- ✓ Cada nova iteração corresponde a uma volta na espiral;
- ✓ Cada volta na espiral (iniciando a partir do centro e avançando para fora) representa uma nova fase do processo.



Prototipagem

Prototipagem Descartável

- ✓ Protótipo (*throw-away*), após sua utilização, são deixados de lado, servindo apenas para consultas.
- ✓ O processo de desenvolvimento do software real é totalmente independente do processo utilizado para a elaboração do protótipo.
- ✓ Normalmente possuem baixa fidelidade, pois são criados para auxiliar a especificação de requisitos e não deveriam integrar o desenho do software.

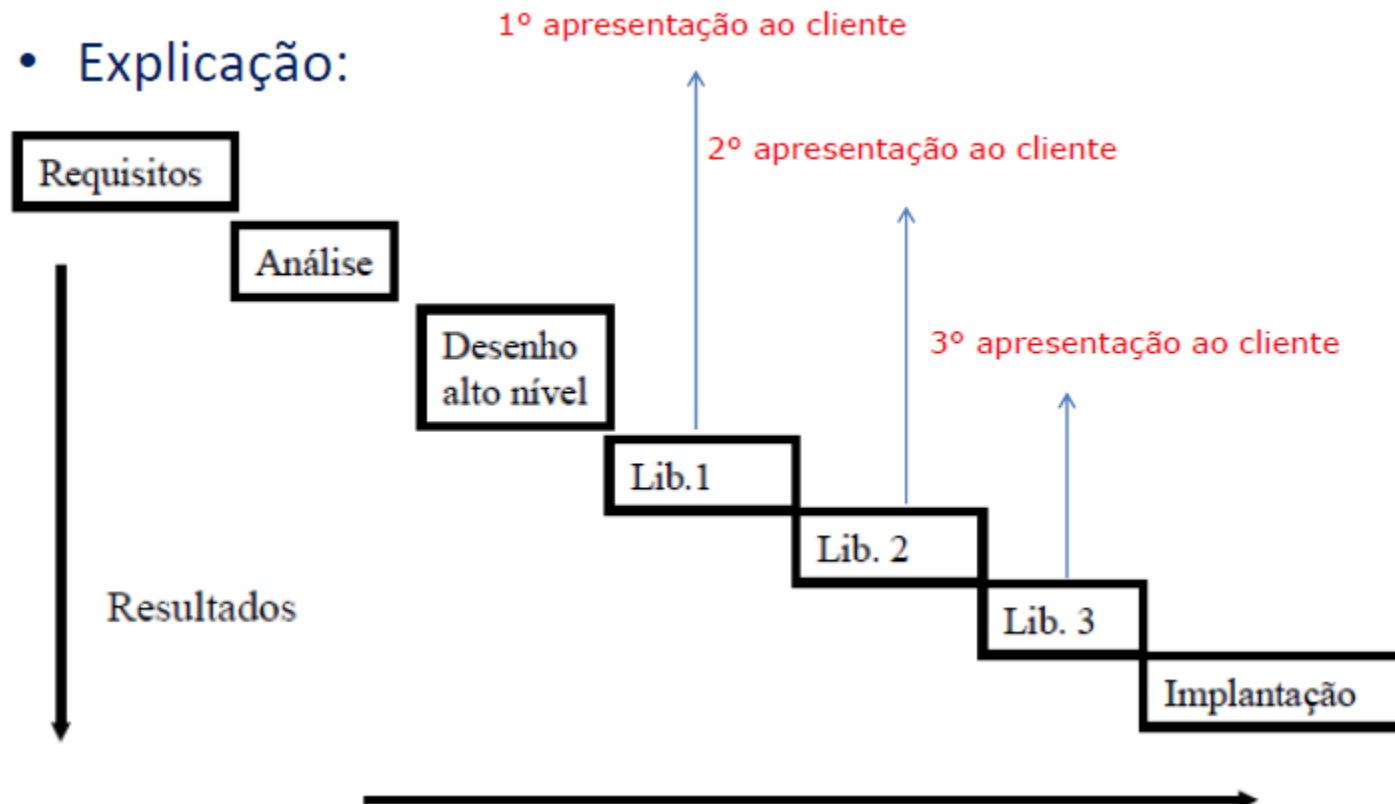
Prototipagem Evolutiva

- ✓ Protótipos evolutivos (*evolutionary*) são criados nas fases iniciais do projeto e refinados ao longo do decorrer do processo de desenvolvimento do software.
- ✓ São interpretados como liberações. Incrementos de funcionalidade são incorporados ao protótipo, com fidelidade gradualmente aumentada, se torna o software final.
- ✓ Os processos de desenvolvimento do protótipo e do software real são essencialmente o mesmo.

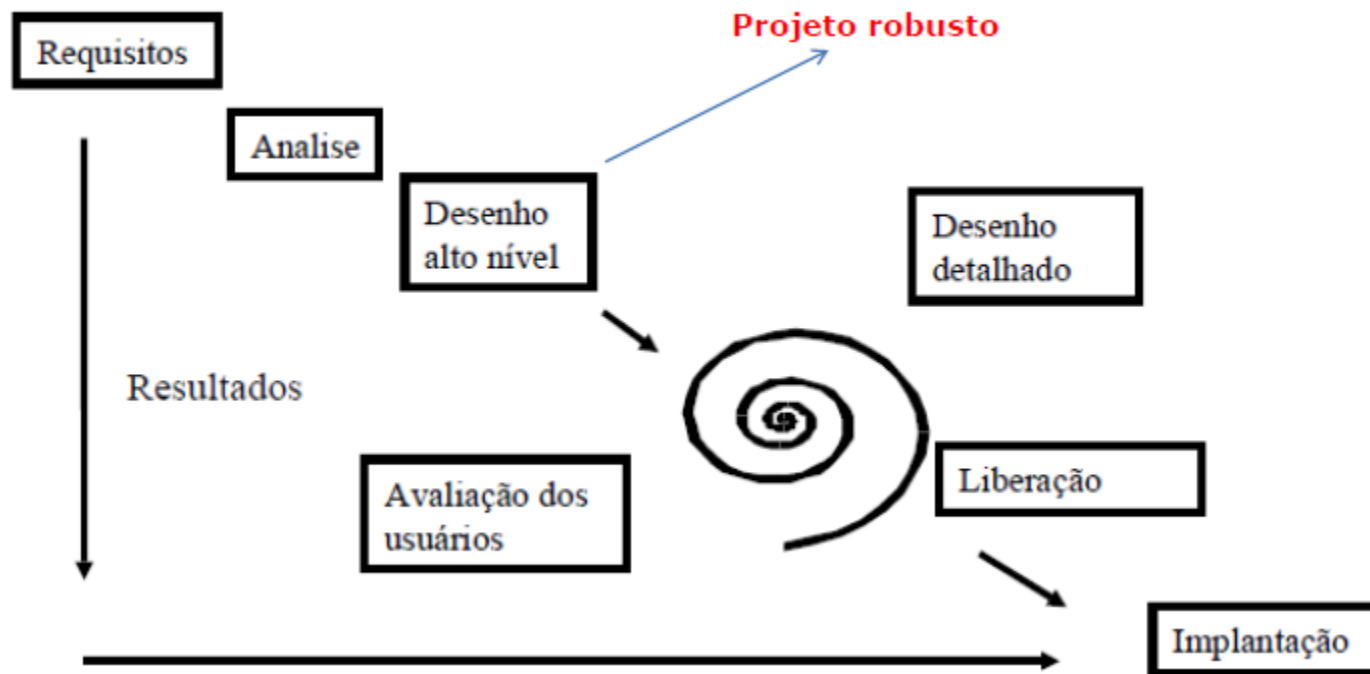


Entrega por Estágios

- Explicação:



Entrega Evolutiva



RAD

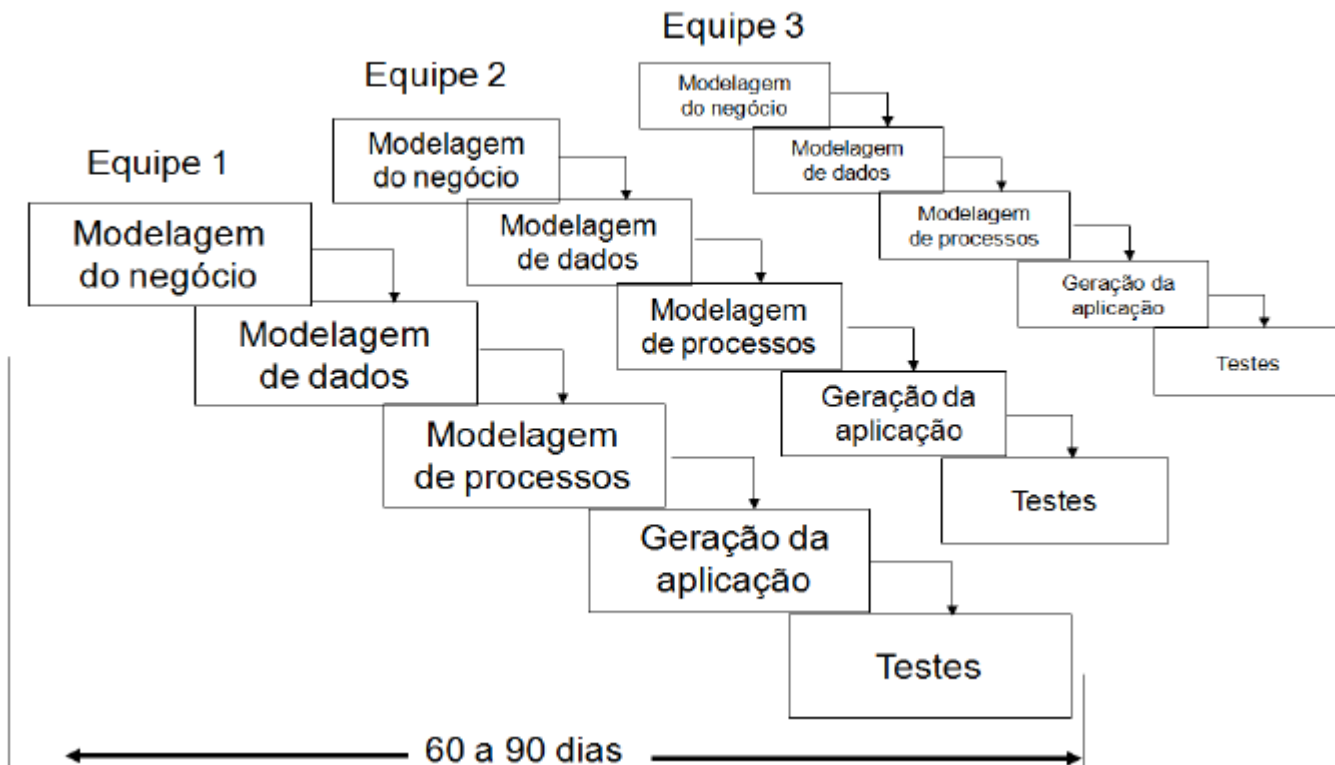
Rapid Application Development (RAD)
ou Desenvolvimento Rápido de Aplicação.

Modelo de processo de desenvolvimento de software iterativo e incremental que enfatiza um ciclo de desenvolvimento extremamente curto (entre 60 e 90 dias).





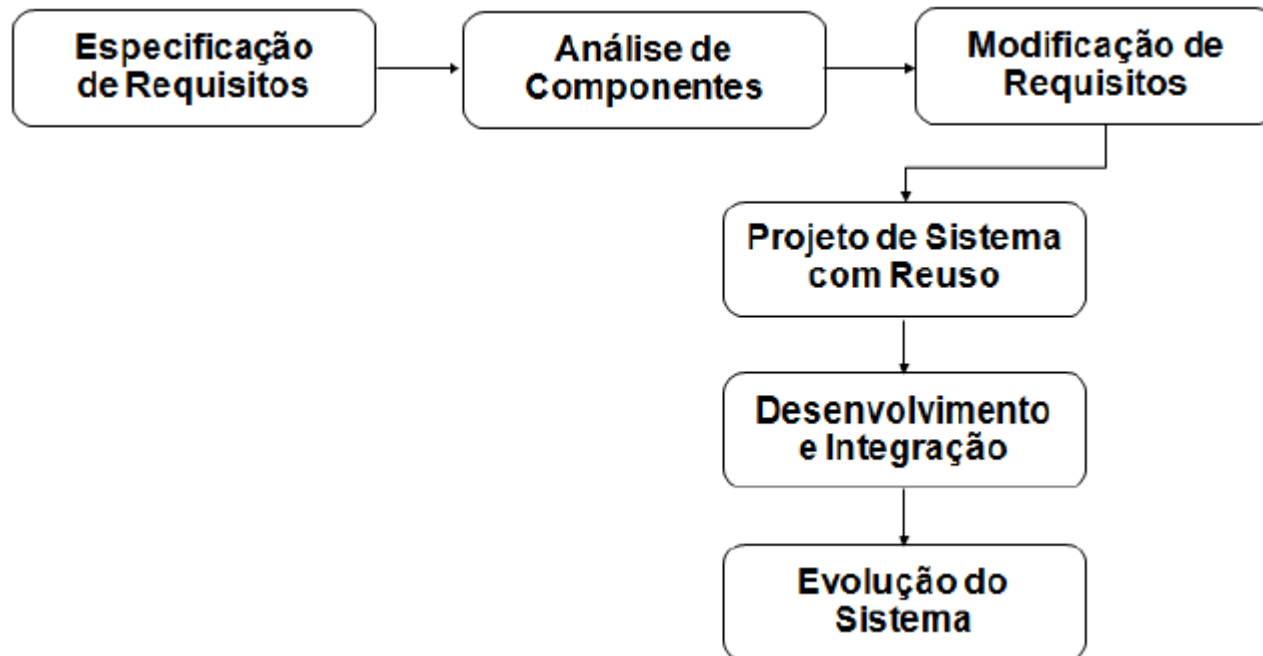
RAD





Orientado a Reuso

São baseados no uso sistemático de componentes/itens pré-existent





Questions





Engenharia de Software Baseada em Componentes





Engenharia de Software baseada em componentes

(CBSE – Component Based Software Engineering) é o desenvolvimento de Software baseado na reutilização de software através de componentes



Componentes de Software é o elemento de software que encapsula uma série de funcionalidades.

Um componente é uma unidade independente, que pode ser utilizado com outros componentes para formar um sistema mais complexo.



Componentes são como caixas-pretas, ou seja, são manipulados com base exclusivamente na sua Definição/Funcionalidade.

A construção de um sistema baseado em reuso de componentes é feita através das conexões entre componentes já existentes e seus conectores ou interfaces, aproveitando assim, suas funcionalidades.





Quais os critérios para
seleção de um modelo de
ciclo de vida?





Critérios de Seleção

1. Usuários
2. Problema
3. Produto
4. Recursos
5. Desenvolvimento

Critérios relacionados aos usuários / equipe

- Experiência dos usuários no domínio da aplicação.
- Facilidade dos usuários em expressar requisitos.
- Experiência da equipe de desenvolvimento no domínio da aplicação.
- Experiência da equipe de desenvolvimento em engenharia de *software*.

Critérios relacionados ao problema

- Grau de maturidade do domínio da aplicação.
- Complexidade do problema.
- Frequência de mudanças nos requisitos.
- Grau de magnitude das mudanças nos requisitos.
- Grau de modularidade do problema.

Critérios relacionados ao produto

- Tamanho da aplicação.
- Grau de complexidade da aplicação.
- Grau de importância dos requisitos de interface.

Critérios relacionados aos recursos

- Disponibilidade de recursos humanos.
- Grau de acesso aos usuários.

Critérios relacionados ao desenvolvimento

- Aplicável à necessidade de entrega de produtos intermediários.
- Grau dos riscos técnicos.
- Paradigma adotado.





Questions





Prova-1
4 pts





Faculdade
IMPACTA
TECNOLOGIA





Processo de Software

- ✓ História
- ✓ Conceitos da Engenharia de Software
- ✓ Processo de Software
- ✓ Modelos de Processo
- **Introdução Métodos Ageis**
- ✓ Introdução Scrum
- ✓ Introdução UP
- ✓ Introdução RUP





Metodos Ágeis





Metodologia - Estudo dos Métodos

Métodos – caminhos usados para
se atingir um objetivo





- Quais são os **problemas** existentes até então que motivaram uma nova “onda” de desenvolvimento de Software?



Motivação

- ✓ Insatisfação
- ✓ Morosidade no desenvolvimento
- ✓ Burocrático
- ✓ Ineficiente
- ✓ Foco no produto / código - Não no projeto
- ✓ Entregas mais rápidas
- ✓ Modelo Iterativo
- ✓ Bastante feedback
- ✓ Sem desperdícios





- ✓ 2001
- ✓ Agilemanifesto.org
- ✓ Uma série de pessoas já vinham desenvolvendo software de maneiras não convencionais





• Signatários

- **Kent Beck**
- Mike Beedle
- Arie van Bennekum
- Alistair Cockburn
- Ward Cunningham
- **Martin Fowler**
- Robert C. Martin
- **Ken Schwaber**
- Dave Thomas

James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Steve Mellor
Jeff Sutherland





Manifesto para o desenvolvimento ágil de software

2001

Estamos descobrindo maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo. Através deste trabalho, passamos a valorizar:

Indivíduos e interação entre eles 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



Princípios por trás do manifesto ágil

- ✓ A 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.





Metodos Ágeis

*“A engenharia de software ágil combina filosofia com um conjunto de princípios de desenvolvimento. A filosofia defende a **satisfação do cliente** e a **entrega de incremental prévio**; **equipes de projetos pequenas** e **altamente motivadas**; métodos informais; artefatos de engenharia de software mínimos e, acima de tudo, **simplicidade no desenvolvimento geral**. Os princípios de desenvolvimento **priorizam a entrega** mais que a análise e projeto (embora essas atividades não sejam desencorajadas); também priorizam a comunicação ativa e contínua entre desenvolvedores e clientes”.*
(Pressman,2011)





*A coisa mais importante sobre os **Métodos Ageis** é que **não existe processo, existem times ageis.***

Os processos descritos tem a ver com ambientes onde equipes podem ser ageis.



"O desenvolvimento ágil é incremental e necessita de práticas ageis:

- ✓ TDD - Test-Driven Development, ou Desenvolvimento Guiado por Testes, consiste numa técnica de desenvolvimento de software onde primeiro são criados os testes e somente depois é escrito o código necessário para passar por eles. Permite fazer testes contínuos
- ✓ Planejamento Incremental: planejar em etapas, Os requisitos de cada incremento (releases pequenos) podem ser registrados em cartões, determina-se a prioridade e o tempo com o cliente o tempo
- ✓ Refatoração: melhorando o código e tornando-o mais fácil de manter constantemente;



- ✓ Integração contínua: quando o incremento está pronto, ele é integrado ao sistema como um todo, ou seja, isto é feito diariamente.
- ✓ Equipe ser auto gerenciável, e tem um líder, como facilitador.
- ✓ Equipes pequenas

Mike Cohn afirma que equipes pequenas possuem menos ociosidade social, melhoram a interação construtiva, menor tempo na coordenação, ninguém fica para trás, pois todos apreendem em conjunto, há maior satisfação entre os membros do grupo e é menos provável que ocorra excesso de especialização, pois todos devem conhecer o projeto.





Três Pontos de Atenção

- ✓ O modelo é proposto por programadores – focado em implementação. Desenho e Usabilidade em segundo plano
- ✓ O Produto do Desenvolvimento é “quebrado” em partes muito pequenas – gerando risco de integração, dificultando a interação do usuário
- ✓ Tempo muito curto para as entregas, dificultando testes e usabilidade

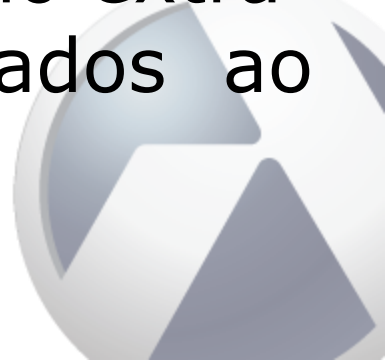


Solução

- ✓ Executar Testes de Usabilidade, simulando testes do usuário, antes mesmo de saber todos os detalhes do produto final.
- ✓ Usuários com muito conhecimento dos processos e fluxos de negócios



Problemas dos Métodos Ágeis

- ✓ Dificuldade de manter o interesse do usuário
 - ✓ Equipes podem ter dificuldade com a intensidade dos trabalhos
 - ✓ Dificuldade de Priorizar mudanças com tantos stakeholders envolvidos
 - ✓ Manter a simplicidade requer trabalho extra
 - ✓ Problemas com contratos relacionados ao escopo
- 

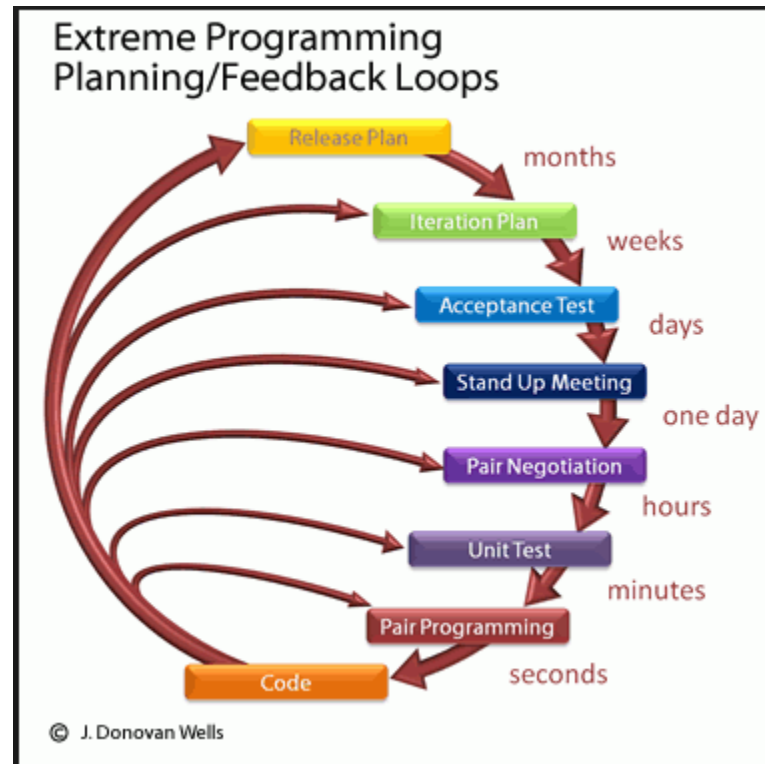


Metodos Ágeis





Extreme Programming





“Trata-se de uma metodologia ágil para equipes pequenas e médias desenvolvendo software com requisitos vagos e constante mudança”
Kent Beck

Kent Beck é um [engenheiro de software](#) americano criador do [Extreme Programming](#) e [Test Driven Development](#). Beck foi um dos 17 signatários originais do [Agile Manifesto](#) em 2001.¹



✓ **Comunicação**

Métodos para rapidamente construir e disseminar conhecimento

✓ **Simplicidade**

XP encoraja que você comece, sempre, pela solução mais simples que funcione

✓ **Feedback**

Do cliente, do sistema e da equipe

✓ **Coragem**

Design simples, refatoração...

✓ **Respeito**

Respeito da equipe, do cliente, dos usuários..



Práticas XP

- ✓ Time Coeso (Whole Team)
- ✓ Testes de Aceitação (Customer Tests)
- ✓ Ritmo Sustentável (Sustainable Tests)
- ✓ Reuniões em pé (Stand-up Meeting)
- ✓ Posse Coletiva (Collective Ownership)
 - O código fonte não tem dono
- ✓ Programação em Pares (Pair Programming)



Práticas XP

Jogo de Planejamento (Planning Game):

O desenvolvimento é feito em iterações semanais. No início da semana, desenvolvedores e clientes reúnem-se para priorizar as funcionalidades.

Essa reunião recebe o nome de Jogo de Planejamento



Práticas XP

Pequenas Versões (Small Releases):

A liberação de pequenas versões funcionais do projeto auxilia muito no processo de aceitação por parte do cliente, que já pode testar uma parte do sistema que está comprando



Práticas XP

Metáfora (Metaphor):

Procura facilitar a comunicação com o cliente, entendendo a realidade dele - História contada para facilitar entendimento a cerca da funcionalidades do sistema



Práticas XP

Projeto Simples (Simple Design):

Se na primeira versão apenas o usuário “teste” pode entrar no sistema com a senha “123” então somente isso deve ser feito.

Não considerar

- ✓ sistemas de autenticação
- ✓ restrições de acesso



Práticas XP

Padrões de Codificação (Coding Standards):

A equipe de desenvolvimento precisa estabelecer regras para programar e todos devem seguir estas regras



Práticas XP

Desenvolvimento Orientado a Testes (Test Driven Development):

Primeiro crie os testes unitários (unit tests) e depois crie o código para que os testes funcionem



Práticas XP

Refatoração (Refactoring):

Mudar o Design sem mudar a funcionalidade



Práticas XP

Integração Contínua (Continuous Integration):

Assim que termina o desenvolvimento de um módulo, este já é integrado



Práticas XP

On-Site Team

Equipe precisa estar dentre das instalações do clientes, ou seja, junto com o cliente



Práticas XP

Cenários

- ✓ Requisitos expressos como cenários e histórias de usuários (user story)
- ✓ Escritos em cartões (cards)
- ✓ Escolhas são feitas com base nos "cards" e nos prazos.



User Story

- ✓ “User story” é similar ao caso de uso
- ✓ Usados para estimar tempo
- ✓ Usados no lugar de requisitos muito extensos
- ✓ São escritos pelos usuários, demonstrando que o sistema deve fazer por eles
- ✓ Tem cerca de três sentenças, com a terminologia do próprio usuário, portanto de negócio
- ✓ Auxiliam na criação de testes de aceitação



User Story

- ✓ A diferença entre especificação de requisitos e “use story” está nos detalhes
- ✓ O detalhe deve ser apenas o suficiente para a estimativa de tempo para a implementação
- ✓ Quando se inicia a implementação / codificação, obtem-se junto ao usuários os detalhes necessários.





Questions



Pós-Graduação

Engenharia de Software



F a c u l d a d e
IMPACTA
T E C N O L O G I A

Obrigada

Marta Fuzioka

mrtfuzioka@uol.com.br