Engenharia de Software I

Métodos Ágeis

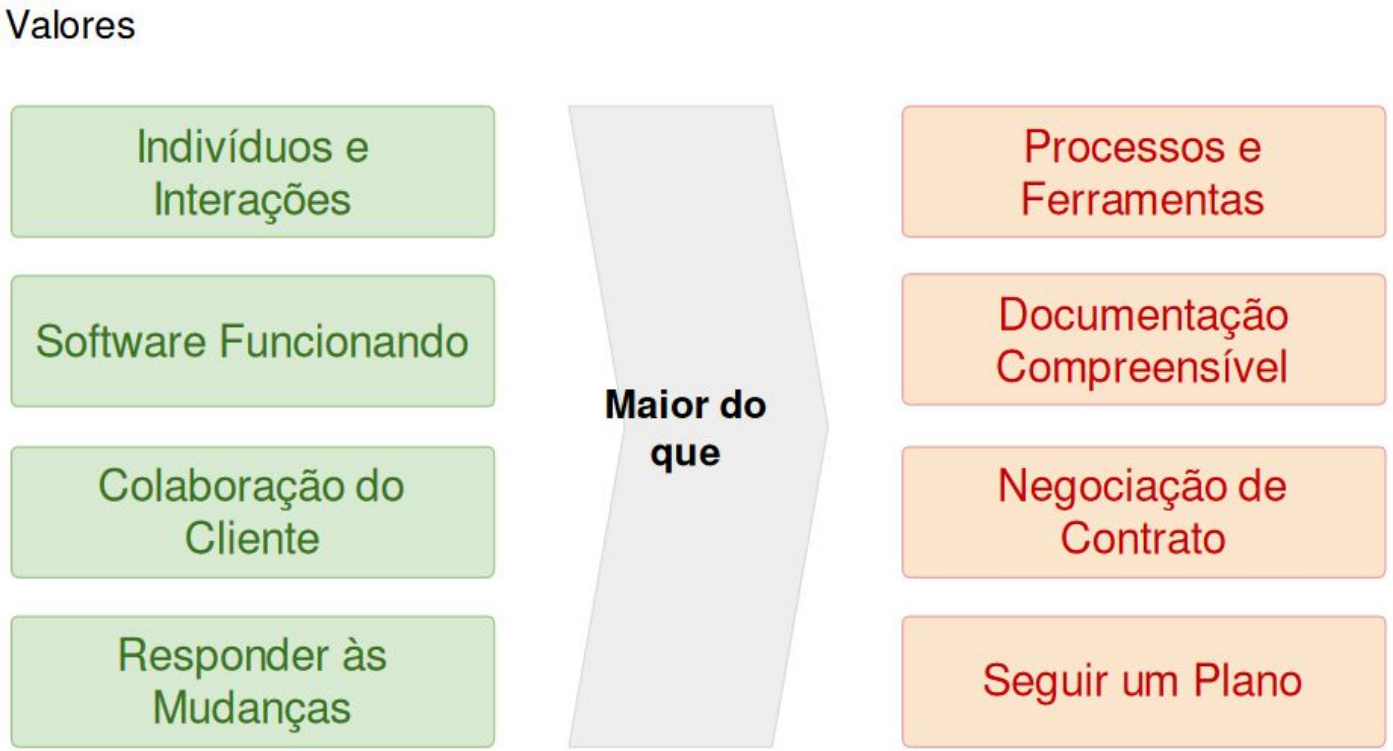
Métodos Ágeis de Desenvolvimento de Software

- Movimento iniciado por programadores experientes e consultores em desenvolvimento de software.
- Métodos Ágeis surgem para suprir esta necessidade rápida de mudança;
- Questionam e se opõe a uma série de mitos/práticas adotadas em abordagens tradicionais de Engenharia de Software e Gerência de Projetos.
- Manifesto Ágil:
    - https://agilemanifesto.org/
    - Assinado por 17 desenvolvedores em Utah em fevereiro/2001.

2

Manifesto Ágil

"Estamos descobrindo maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo"



"Embora os itens à direita sejam importantes, valorizamos mais os que estão na esquerda"

Métodos Ágeis de Desenvolvimento de Software

- 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.

4

Manifesto Ágil

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 freqüencia, na escala de semanas até meses, com preferência aos períodos mais curtos.

Pessoas relacionadas a negócios e desenvolvedores devem **trabalhar em conjunto** e diariamente, durante todo o curso do projeto.

Manifesto Ágil

Princípios

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.

Manifesto Ágil

Princípios

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 **ajusta** e otimiza seu comportamento de acordo.

Principais Métodos Ágeis

- Programação eXtrema (XP)
- Scrum
- Feature-Driven Development (FDD)
- Dynamic Systems Development Method (DSDM)
- Adaptive Software Development (ASD)
- Crystal Clear

8

eXtreme Programming Programming – XP

- Metodologia ágil (leve) mais utilizada na atualidade
- Desenvolvida para:
  - Equipes médias e pequenas (2 a 12 pessoas)
  - Requisitos vagos e em constante evolução
- Possui um conjunto de valores e práticas para nortear o desenvolvimento de software

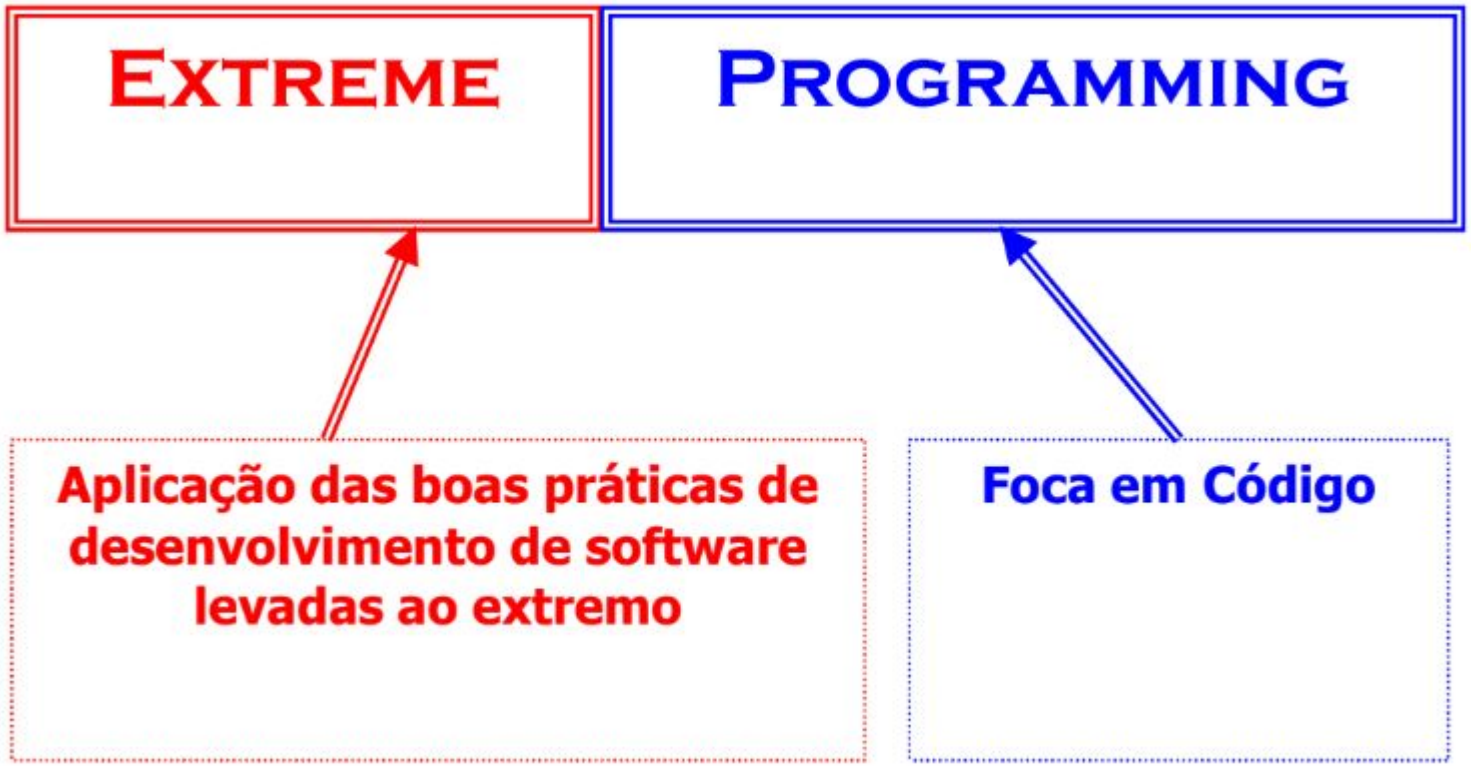
9

O surgimento do XP

- Em meados de 1990, Kent Beck procurou formas mais simples e eficientes de desenvolver software
  - Identificou o que tornava simples e o que dificultava o desenvolvimento de software
- Em Março de 1996, ele iniciou um projeto com novos conceitos que resultaram na metodologia XP - eXtreme Programming
- “Trata-se de uma metodologia ágil para equipes pequenas e médias desenvolvendo software com requisitos vagos e em constante mudança” - Kent Beck

10

XP - Estudo da palavra



Programando ao Extremo

- Levar todas as boas práticas ao Extremo
  - Se testar é bom, vamos testar toda hora!!
  - Se projetar é bom, vamos fazer disso parte do trabalho diário de cada pessoa!
  - Se integrar é bom, vamos integrar a maior quantidade de vezes possível!
  - Se iterações curtas é bom, vamos deixar as iterações realmente curtas!

12

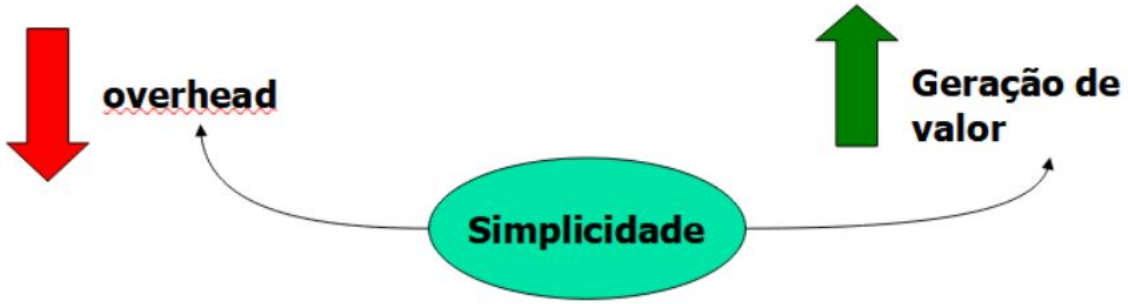
Mudanças sempre ocorrem

- Clientes não sabem o que querem no início
- Desenvolvedores não sabem qual a melhor maneira de fazer o software no início
- Medo da mudança trava o desenvolvimento

13

XP – Princípios Básicos

- Comunicação
  - Os membros da equipe (clientes, gerentes, programadores) devem interagir ao máximo pessoalmente.
  - Devem trabalhar na mesma sala, conversar pessoalmente ou através de chats, etc.
  - Comunicação face a face é a maneira mais rápida de “espalhar” conhecimento
- Simplicidade
  - Projeto do SW é simplificado continuamente
  - Processo adaptado, caso algo não esteja funcionando
  - Pensar sempre : “ O que pode ser feito que seja o mais simples que funciona?”
  - Simplicidade normalmente gera mais valor



14

XP – Princípios Básicos

- Feedback
  - Cliente está sempre participando do desenvolvimento do sistema
  - Testes de unidade e de aceitação fornecem feedback sobre o sistema
  - Oportunidades e problemas são identificados o mais rápido possível
  - Aumenta aprendizado e produtividade
  - Feedback <-> Comunicação
- Coragem
  - Indicar problemas no projeto, parar quando está cansado, pedir ajuda quando necessário
  - Simplificar código que está funcionando, dizer ao cliente que não será possível implementar um requisito
  - Seguir o XP como deve ser

15

XP - Práticas

- Jogo do Planejamento
- Versões Pequenas
- Metáfora
- Projeto Simples
- Desenvolvimento Orientado por Testes
- Testes dos Clientes
- Refatoração
- Programação Pareada
- Propriedade Coletiva do Código
- Integração Contínua e Frequente
- Ritmo Sustentável
- Cliente com os desenvolvedores
- Padrão de Código

16

XP

- Jogo de Planejamento (Planning Game):
  - O desenvolvimento é feito em iterações semanais.
  - No início da semana, desenvolvedores e cliente reúnem-se para priorizar as funcionalidades.
  - Essa reunião recebe o nome de Jogo do Planejamento.
  - Nela, o cliente identifica prioridades e os desenvolvedores as estimam.
  - O cliente é essencial neste processo e assim ele fica sabendo o que está acontecendo e o que vai acontecer no projeto.
  - Como o escopo é reavaliado semanalmente, o projeto é regido por um contrato de escopo negociável, que difere significativamente das formas tradicionais de contratação de projetos de software.
  - Ao final de cada semana, o cliente recebe novas funcionalidades, completamente testadas e prontas para serem postas em produção.
- 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.

17

XP

- Metáfora (Metaphor):
  - Procura facilitar a comunicação com o cliente, entendendo a realidade dele.
  - O conceito de rápido para um cliente de um sistema jurídico é diferente para um programador experiente em controlar comunicação em sistemas em tempo real, como controle de tráfego aéreo.
  - É preciso traduzir as palavras do cliente para o significado que ele espera dentro do projeto.
- Projeto Simples (Simple Design):
  - Simplicidade é um princípio da XP.
  - Um erro comum ao adotar essa prática é a confusão por parte dos programadores de código simples e código fácil.
  - Nem sempre o código mais fácil de ser desenvolvido levará a solução mais simples por parte de projeto.
  - Esse entendimento é fundamental para o bom andamento do XP.
  - Código fácil deve ser identificado e substituído por código simples.

18

XP

- Testes de Aceitação (Customer Tests):
  - São testes construídos pelo cliente e conjunto de analistas e testadores, para aceitar um determinado requisito do sistema.
- Ritmo Sustentável (Sustainable Pace):
  - Trabalhar com qualidade, buscando ter ritmo de trabalho saudável (40 horas/semana, 8 horas/dia), sem horas extras.
  - Horas extras são permitidas quando trouxerem produtividade para a execução do projeto.
  - Outra prática que se verifica neste processo é a prática de trabalho energizado, onde se busca trabalho motivado sempre.
  - Para isto o ambiente de trabalho e a motivação da equipe devem estar sempre em harmonia.
- Reuniões em pé (Stand-up Meeting):
  - Reuniões em pé para não se perder o foco nos assuntos, produzindo reuniões rápidas, apenas abordando tarefas realizadas e tarefas a realizar pela equipe.

19

XP

- Programação em Pares (Pair Programming):
  - É a programação em par/dupla num único computador.
  - Geralmente a dupla é formada por um iniciante na linguagem e outra pessoa funcionando como um instrutor.
  - Como é apenas um computador, o novato é que fica à frente fazendo a codificação, e o instrutor acompanha ajudando a desenvolver suas habilidades.
  - Desta forma o programa sempre é revisto por duas pessoas, evitando e diminuindo assim a possibilidade de defeitos.
  - Com isto busca-se sempre a evolução da equipe, melhorando a qualidade do código fonte gerado.

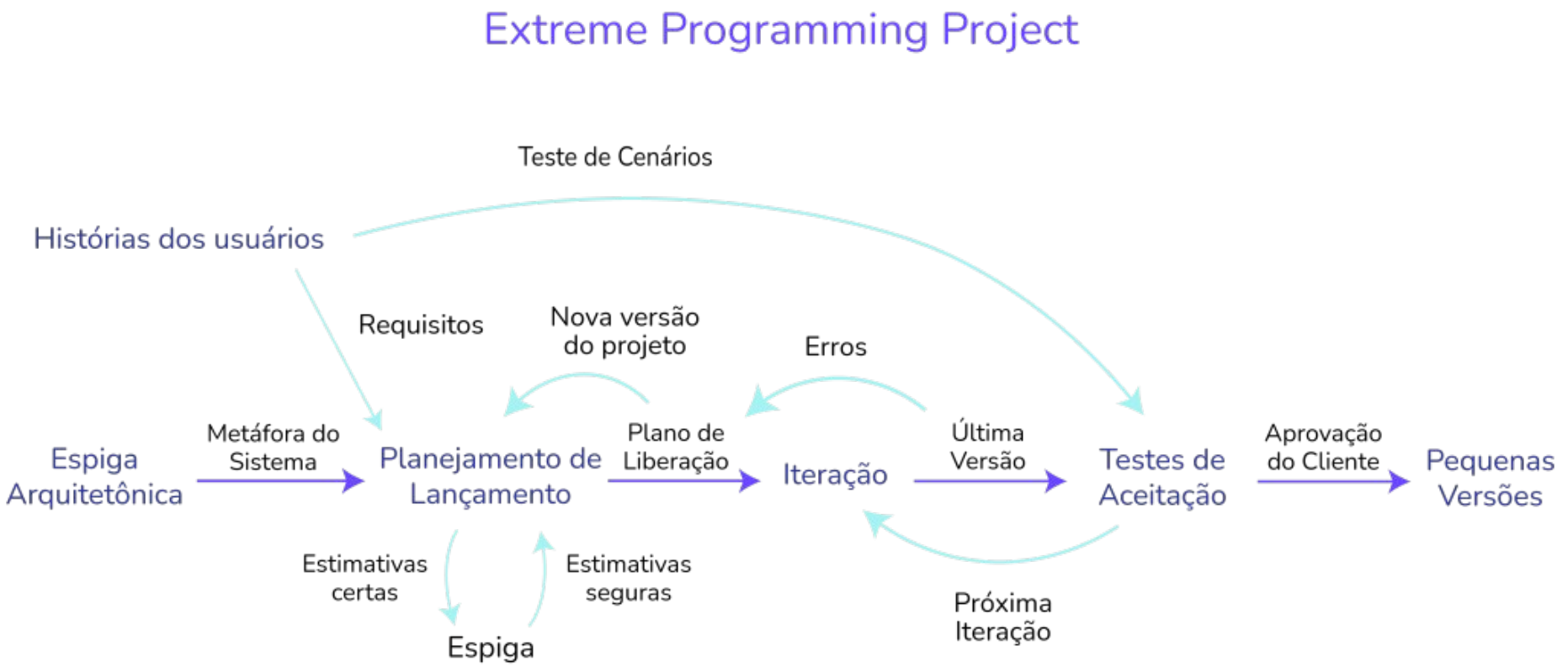
20

XP

- Padrões de Codificação (Coding Standards):
  - A equipe de desenvolvimento precisa estabelecer regras para programar e todos devem seguir estas regras.
  - Desta forma parecerá que todo o código fonte foi editado pela mesma pessoa, mesmo quando a equipe possui 10 ou 100 membros.
- Refatoração (Refactoring):
  - É um processo que permite a melhoria contínua da programação, com o mínimo de introdução de erros e mantendo a compatibilidade com o código já existente.
  - Refabricar melhora a clareza (leitura) do código, divide-o em módulos mais coesos e de maior reaproveitamento, evitando a duplicação de código-fonte;
- Integração Contínua (Continuous Integration):
  - Sempre que produzir uma nova funcionalidade, nunca esperar uma semana para integrar à versão atual do sistema.
  - Isto só aumenta a possibilidade de conflitos e a possibilidade de erros no código fonte.
  - Integrar de forma contínua permite saber o status real da programação.

21

Metodologia de Projeto - XP



22

The Rules of Extreme Programming

The Rules of Extreme Programming

Planning

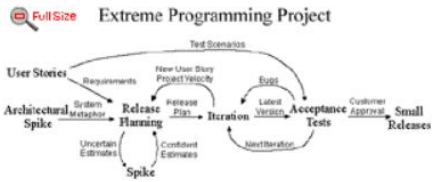
- User stories are written.
- Release planning creates the release schedule.
- Make frequent small releases.
- The project is divided into iterations.
- Iteration planning starts each iteration.

Managing

- Give the team a dedicated open work space.
- Set a sustainable pace.
- A stand up meeting starts each day.
- The Project Velocity is measured.
- Move people around.
- Fix XP when it breaks.

Designing

- Simplicity.
- Choose a system metaphor.
- Use CRC cards for design sessions.
- Create spike solutions to reduce risk.
- No functionality is added early.
- Refactor whenever and wherever possible.



Coding

- The customer is always available.
- Code must be written to agreed standards.
- Code the unit test first.
- All production code is pair programmed.
- Only one pair integrates code at a time.
- Integrate often.
- Set up a dedicated integration computer.
- Use collective ownership.

Testing

- All code must have unit tests.
- All code must pass all unit tests before it can be released.
- When a bug is found tests are created.
- Acceptance tests are run often and the score is published.

Let's review the values of Extreme Programming (XP) next.

ExtremeProgramming.org home | XP Map | XP Values | Test framework | About the Author

23

Feature Driven Development - FDD (Desenvolvimento Guiado por Funcionalidades)

- Metodologia ágil para gerenciamento e desenvolvimento de software
- Ela combina as melhores práticas do gerenciamento ágil de projetos com uma
- abordagem completa para Engenharia de Software orientada por objetos
- O lema da FDD é: "Resultados frequentes, tangíveis e funcionais.“
- Enfatiza a produção de software de qualidade
- Realiza trabalho significativo desde o início, antes de tornar-se altamente iterativa

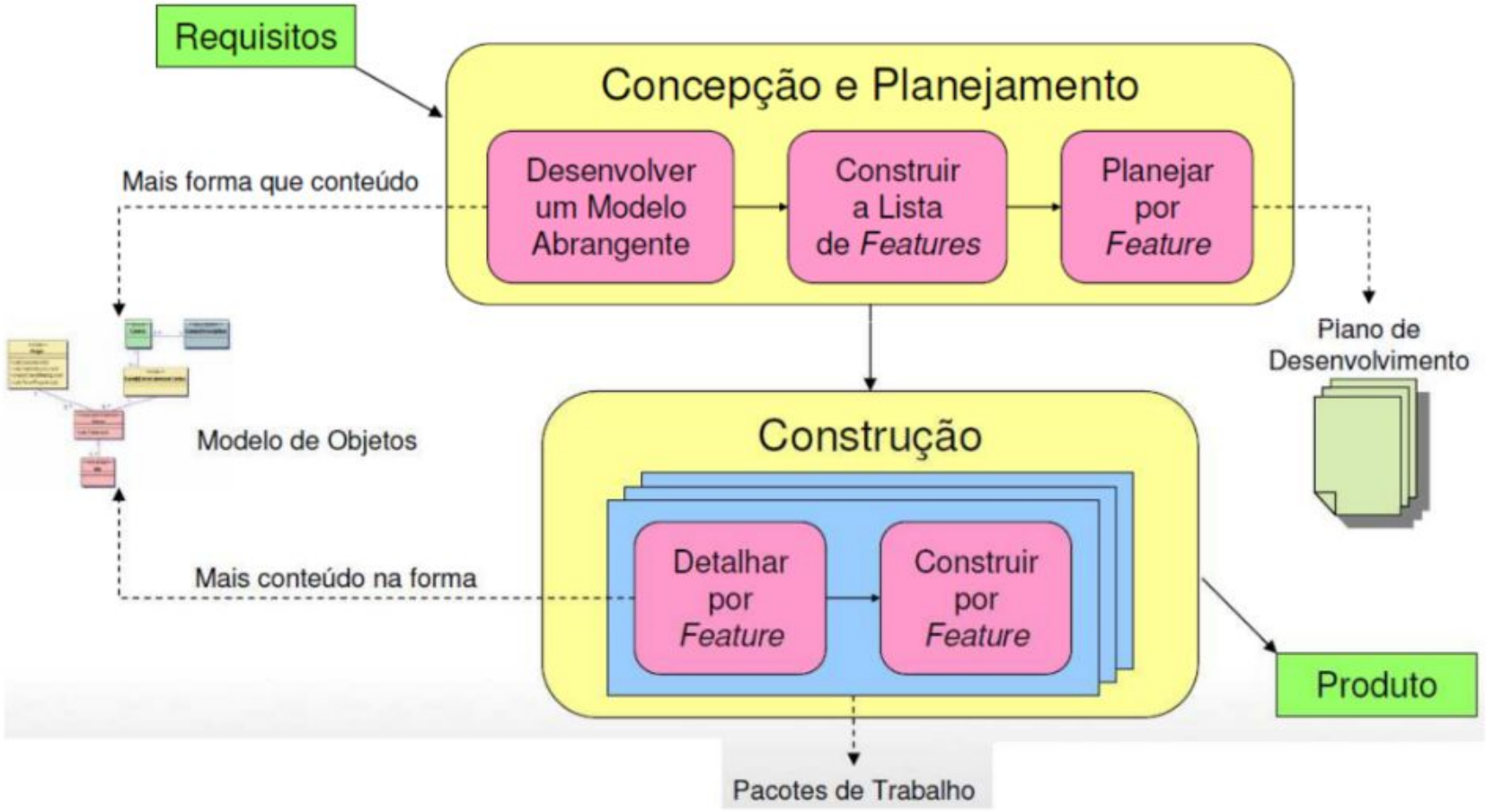
24

FDD - O que é uma feature?

- Característica ou funcionalidade
- Pequena o suficiente para ser implementada no máximo em 2 semanas
- Oferece valor para o cliente
- Mapeia passos em uma atividade de negócio
  - Pode ser um passo de um caso de uso
  - Às vezes pode ser o próprio caso de uso
- Conceito muito próximo ao de um requisito funcional
- Modelo: <ação> <resultado> <objeto>
  - Calcular o total de uma venda
  - Autorizar uma transação com cartão de um cliente

25

FDD


Requisitos
Concepção e Planejamento
Desenvolver um Modelo Abrangente
Construir a Lista de Features
Planejar por Feature
Mais forma que conteúdo
Plano de Desenvolvimento
Modelo de Objetos
Construção
Detalhar por Feature
Construir por Feature
Mais conteúdo na forma
Produto
Pacotes de Trabalho

FDD

- A FDD é uma metodologia muito objetiva.Possui apenas duas fases:
  - Concepção e Planejamento: Pensar um pouco antes de fazer (tipicamente de 1 a 2 semanas)
  - Construção: Fazer de forma iterativa (tipicamente em iterações de 2 semanas)
- Os cinco processos são bem definidos e integrados:
  - DMA (Desenvolver um Modelo Abrangente): Análise orientada por objetos
  - CLF (Construir a Lista de Funcionalidades): Decomposição funcional
  - PPF (Planejar por Funcionalidade): Planejamento incremental
  - DPF (Detalhar por Funcionalidade): Desenho (Projeto) orientado por objetos
  - CPF (Construir por Funcionalidade): Programação e teste orientados por objetos

27

FDD

- **DMA - Desenvolver um Modelo Abrangente:** pode envolver desenvolvimento de requisitos, análise orientada por objetos, modelagem lógica de dados e outras técnicas para entendimento do domínio de negócio em questão. O resultado é um modelo de objetos (e/ou de dados) de alto nível, que guiará a equipe durante os ciclos de construção.

- **CLF - Construir uma Lista de Funcionalidades:** decomposição funcional do modelo do domínio, em três camadas típicas: áreas de negócio, atividades de negócio e passos automatizados da atividade (funcionalidades). O resultado é uma hierarquia de funcionalidades que representa o produto a ser construído (também chamado de product backlog, ou lista de espera do produto).

FDD

- **PPF - Planejar por Funcionalidade Planejar por Funcionalidade:** abrange a estimativa de complexidade e dependência : abrange a estimativa de complexidade e dependência das funcionalidades, também levando em consideração a prioridade e valor para o negócio/cliente. O resultado é um plano de desenvolvimento, com os pacotes de trabalho na seqüência apropriada para a construção.
- **DPF - Detalhar por Funcionalidade:** já dentro de uma iteração de construção, a equipe detalha os requisitos e outros artefatos para a codificação de cada funcionalidade, incluindo os testes. O projeto para as funcionalidades é inspecionado. O resultado é o modelo de domínio mais detalhado e os esqueletos de código prontos para serem preenchidos.
- **CPF - Construir por Funcionalidade:** cada esqueleto de código é preenchido, testado e inspecionado. O resultado é um incremento do produto integrado ao repositório principal de código, com qualidade e potencial para ser usado pelo cliente/usuário.

29

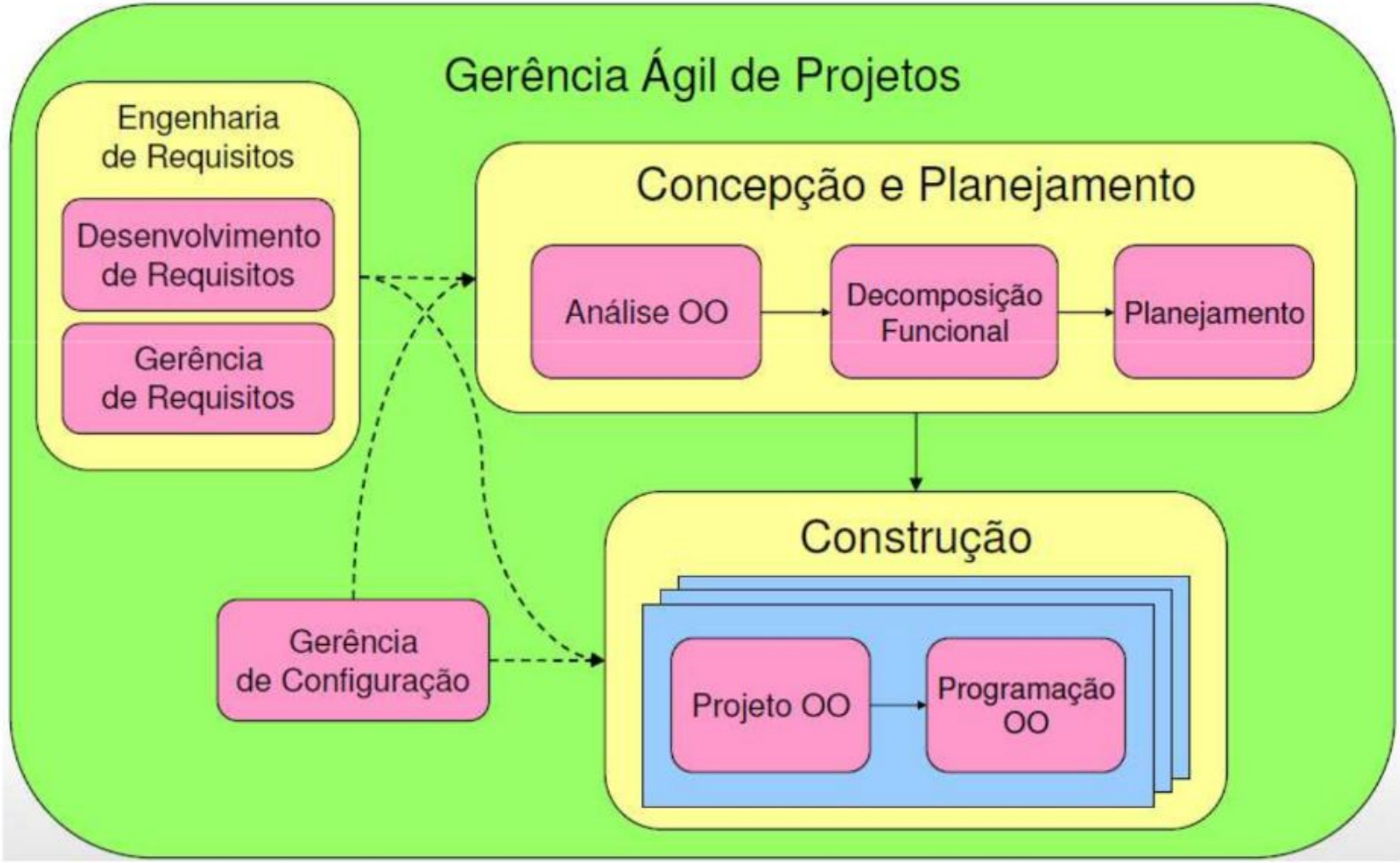
FDD

O Porquê de Cada Processo:

- Desenvolver um Modelo Abrangente
  - Modelagem dos Processos de Negócio (BPM)
  - Análise Orientada por Objetos (OOA)
- Construir a Lista de Features
  - Decomposição Funcional
- Planejar por Feature
  - Plano de Desenvolvimento
  - Prioridade, Dependência, Distribuição de Trabalho
- Detalhar por Feature
  - Projeto OO (OOD), Estudo Detalhado
- Construir por Feature
  - Programação OO (OOP)
  - Inspeção, Testes, Integração

30

FDD – Disciplinas envolvidas



31

FDD

- Cargos e Responsabilidades:
  - Gerente de projeto (Project Manager)
  - Arquiteto líder (Chief architect)
  - Gerente de desenvolvimento (Development Manager)
  - Programador líder (Chief programmer)
  - Proprietário de classe (Class owner)
  - Especialista do domínio (Domain experts) – Gerente do domínio (Domain manager)
- De apoio
  - Gerente de versão (Release manager)
  - Guru de linguagem (Language lawyer/language guru)
  - Engenheiro de construção (Build engineer)
  - “Ferramenteiro” (Toolsmith)
  - Administrador de sistemas (System Administrator)
- Adicionais
  - Testadores (Testers)
  - Instaladores (Deployers)
  - Escritores técnicos (Technical writes)

32

FDD – Boas práticas Boas práticas

- Modelagem de objetos de domínio (Domain Object Modeling)
  - Exploração explicação do problema do domínio
  - Resulta em um arcabouço
- Desenvolver por funcionalidade (Developing by feature)
  - Desenvolvimento e acompanhamento do progresso através de da lista de funcionalidades.
- Proprietários de classes individuais (Individual class ownership)
  - Cada classe possui um único desenvolvedor responsável
- Equipe de funcionalidades (Feature teams)
  - Formação de equipes pequenas e dinâmicas.
  - Inspeção (Inspection)
  - Uso dos melhores métodos conhecidos de detecção de erros.
- Releases freqüentes (Regular Builds)
  - Garantir que existe um sistema sempre disponível e demonstrável.
- Administração de Configuração (Configuration Manager)

33

FDD

Pontos positivos

- Foco nos recursos
- Previsibilidade e controle
- Gerência de risco
- Colaboração de comunicação

Pontos negativos

- Problemas com Baixa definição de recursos
- Não adaptado a constantes mudanças
- Foco excessivo nas funcionalidades

34