PECE – POLI – USP
MBA – Tecnologia de Software

Aula 05
Metodologias Ágeis

Prof. Dr. Rogério Rossi
2021

Roteiro

- Agile Methods - Introdução
- Agile Methods
  - SCRUM
  - XP (*eXtreme Programming*)
  - FDD (*Feature-Driven Development*)
  - LD (*Lean Software Development*)
  - Apêndices
    - DSDM (*Dynamic System Development Method*)
    - ASD (*Adaptive Software Development*)
    - Crystal Clear

Agile Methods - Introdução

"**Agile Software Development: the business of innovation**"
Highsmith, J., Cockburn, A. (2001). IEEE Computer

"**Agile Software Development: the people factor**"
Cockburn, A., Highsmith, J. (2001). IEEE Computer

"**Get ready for Agile methods, with care**"
Boehm, B. (2002). IEEE Computer

Agile Methods - Introdução

Agile Software Development: the business of innovation

(Highsmith & Cockburn, 2001)

Agile Software Development approaches:

- Extreme Programming (XP)
- Crystal Methods
- Lean Development
- SCRUM
- Adaptive Software Development (ASD)

Agile Methods - Introdução

Agile Software Development: the business of innovation

(Highsmith & Cockburn, 2001)

The Agile methods stress two concepts:

1. **The unforgiving honesty of working code**

2. **The effectiveness of people working together with goodwill**

Agile Methods - Introdução

Agile Software Development: the business of innovation

(Highsmith & Cockburn, 2001)

February, 2001 – people representing agile methods

SCRUM, XP, DSDM, ASD, Crystal, FDD and others

sympathetic to the need for alternative

software development methods signing the

Manifesto for Agile Software Development

Agile Methods - Introdução

Agile Software Development: the business of innovation

(Highsmith & Cockburn, 2001)

Agile Manifesto

- ***Individuals and interactions*** *over* ***processes and tools***
- ***Working software*** *over* ***comprehensive documentation***
- ***Customer collaboration*** *over* ***contract negotiation***
- ***Responding to change*** *over* ***following a plan***

Agile Methods - Introdução

Agile Software Development: the business of innovation

(Highsmith & Cockburn, 2001)

- **XP** advocates pair programming for feedback,
- **DSDM** features short-cycle using prototyping.
- **Crystal** and **ASD** advocate end-of-iteration process and team reviews.
- **ASD** and **SCRUM** use end-of-iteration reviews with customer focus groups.

Agile Methods - Introdução

Agile Software Development: the people factor

(Cockburn & Highsmith, 2001)

A dominant idea in **agile development** is that the team can be more effective in **responding to change** if it can

- *reduce the cost of moving information between people, and*
- *reduce the elapsed time between making a decision to seeing the consequences of that decision.*

Agile Methods - Introdução

Agile Software Development: the people factor

(Cockburn & Highsmith, 2001)

To **reduce the cost of moving information between people**, the **agile team** works to

- *place people physically closer,*
- *replace documents with talking in person and at whiteboards, and*
- *improve the team's amicability—its sense of community and morale— so that people are more inclined to relay valuable information quickly*

Agile Methods - Introdução

Agile Software Development: the people factor

(Cockburn & Highsmith, 2001)

To reduce the **time from decision to feedback**, the **agile team**

- *makes user experts available to the team or, even better, part of the team*

and

- *works incrementally.*

Agile Methods - Introdução

Get ready for Agile methods, with care

(Boehm, 2002)

Real-world examples argue for and against agile methods

Agile Methods - Introdução

Get ready for Agile methods, with care

(Boehm, 2002)

Comparing the methods: agile and plan-driven approaches

DEVELOPERS

“There are only so many Kent Becks in the world to lead the team. All of the agile methods put a premium on having premium people….”

Agile Methods - Introdução

Get ready for Agile methods, with care

(Boehm, 2002)

Comparing the methods: agile and plan-driven approaches

CUSTOMERS

- **Agile methods** work best when such customers operate in dedicated mode with the development team
- **plan-driven methods** reduce via documentation and use of architecture review boards to compensate for onsite customer shortfalls

Agile Methods - Introdução

Get ready for Agile methods, with care

(Boehm, 2002)

Comparing the methods: agile and plan-driven approaches

REQUIREMENTS

- **Agile approaches** "are most applicable to turbulent, highchange environments."
- **Plan-driven methods** work best when developers can determine the requirements in advance— including via prototyping—and when the requirements remain relatively stable

Agile Methods - Introdução

Get ready for Agile methods, with care

(Boehm, 2002)

Comparing the methods: agile and plan-driven approaches

ARCHITECTURE

- “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.” “Early” and “continuous” are reasonably compatible goals for small systems developed by great people.
- early results in large systems can lead to major rework when the architecture doesn’t scale up.

Agile Methods - Introdução

Get ready for Agile methods, with care

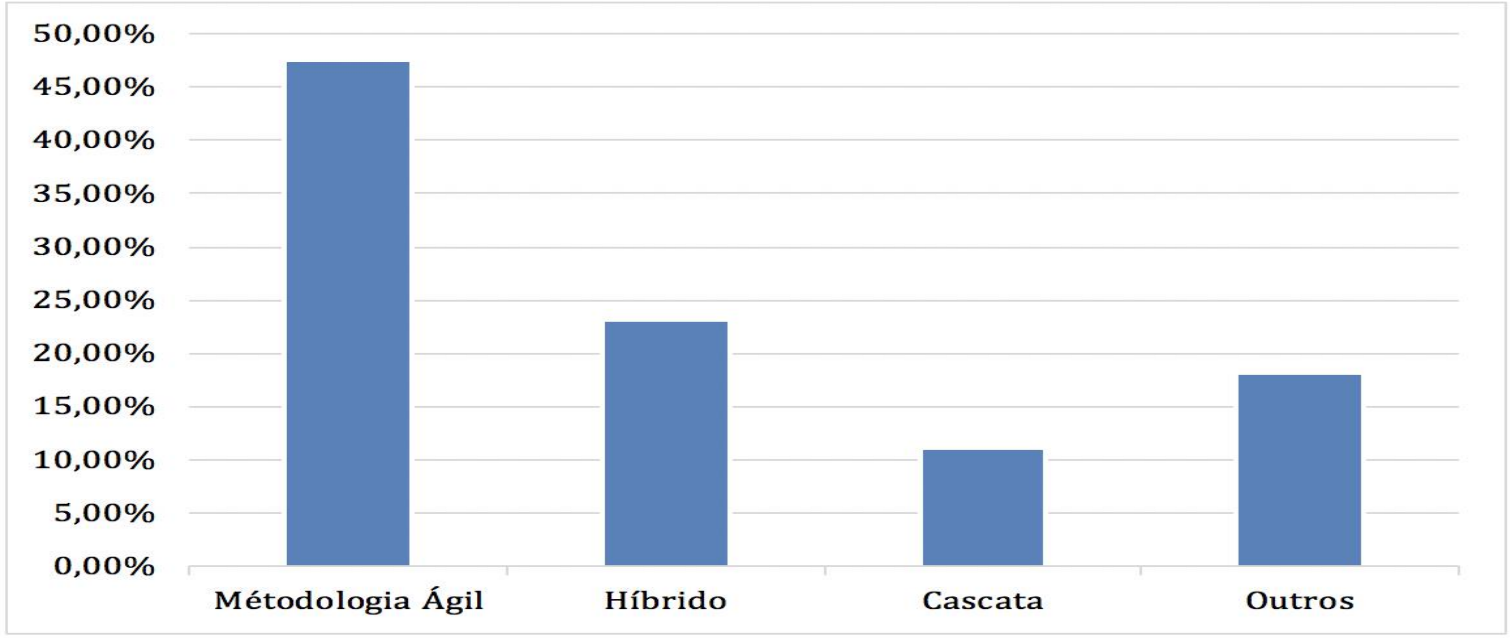
(Boehm, 2002)

Table 1. Home ground for agile and plan-driven methods.

Home-ground area	Agile methods	Plan-driven methods
Developers	Agile, knowledgeable, collocated, and collaborative	Plan-oriented; adequate skills; access to external knowledge
Customers	Dedicated, knowledgeable, collocated, collaborative, representative, and empowered	Access to knowledgeable, collaborative, representative, and empowered customers
Requirements	Largely emergent; rapid change	Knowable early; largely stable
Architecture	Designed for current requirements	Designed for current and foreseeable requirements
Refactoring	Inexpensive	Expensive
Size	Smaller teams and products	Larger teams and products
Primary objective	Rapid value	High assurance

Agile Methods - Introdução

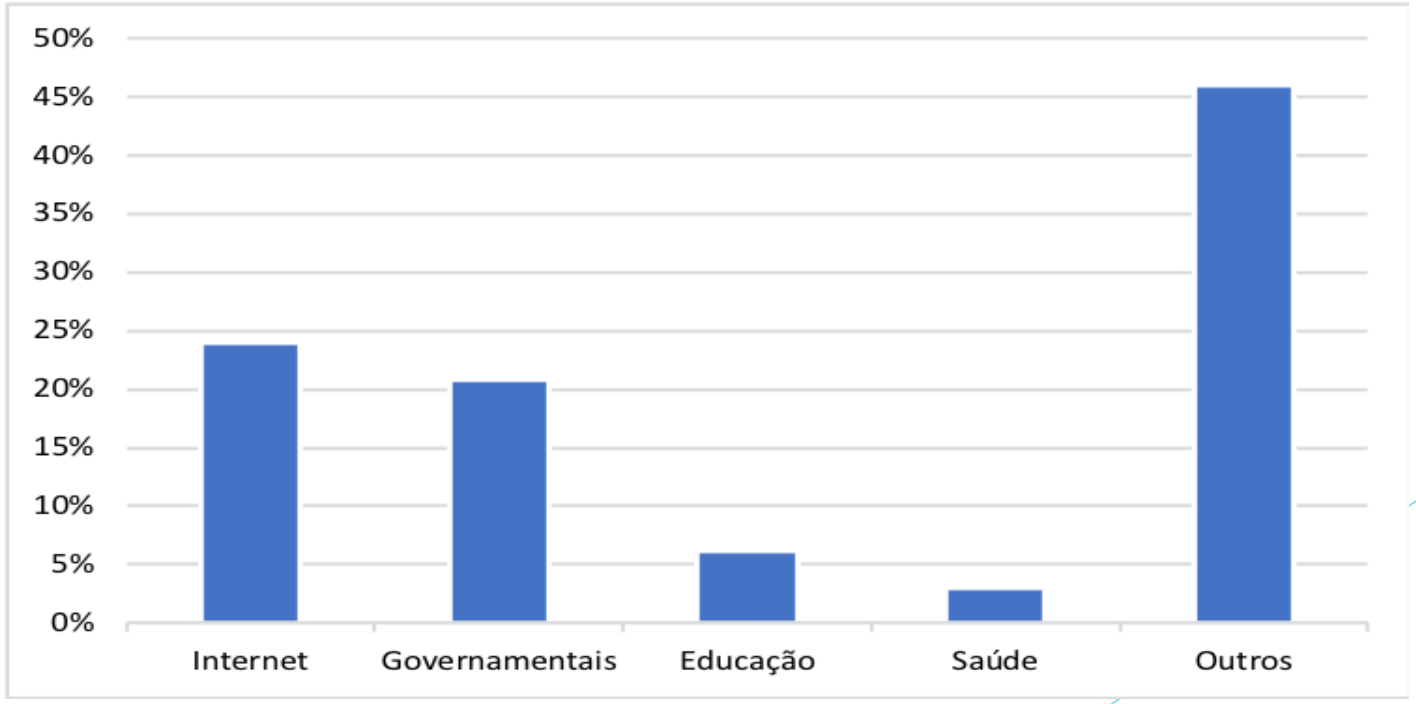
Choudhary & Rakesh (2016), realizou uma **pesquisa com noventa e nove engenheiros de software**, verificou-se que:
47,5% destes trabalham atualmente com **Métodos de Desenvolvimento Ágil**, e **60%** utilizam o **SCRUM** como método de desenvolvimento.



Agile Methods - Introdução

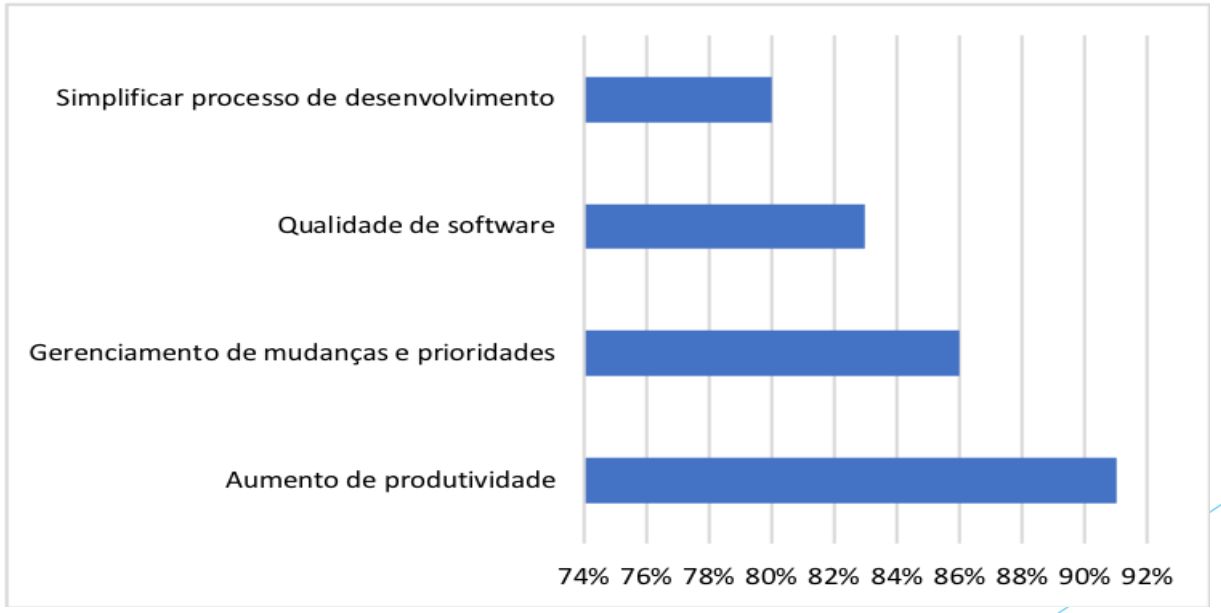
Melo et al. (2013), realizaram uma pesquisa com **471 profissionais brasileiros** de diferentes segmentos da indústria e que utilizam algum **Método Ágil no desenvolvimento de software**.

Empresas de diversos segmentos, tais como: serviços prestados via Internet (24%), governamentais (20,8%), educação (6,2%), saúde (3%) e outros (46%).



Agile Methods - Introdução

Da pesquisa realizada por Melo et al. (2013), dos profissionais participantes:
91% revelaram que o principal objetivo mediante a adoção de uma **Metodologia Ágil** é o **aumento de produtividade das equipes**,
86% buscam **capacidade de melhorar o gerenciamento de mudanças e prioridades,**
83% buscam **aumentar a qualidade de software,** e
80% desejam **simplificar o processo de desenvolvimento de software.**



Agile Methods - Introdução

Segundo Choudhary & Rakesh (2016), as **metodologias ágeis** também apresentem **algumas desvantagens**.

A **primeira** é referente à **falta de artefatos que descrevem os requisitos** de um determinado sistema baseado em software, que apesar de acelerar o desenvolvimento **prejudica a manutenção e alteração do software**.

Segunda trata de um princípio que o Manifesto Ágil julga imprescindível, que é a **participação do usuário no desenvolvimento do software**, segundo ele, o **sucesso do projeto** está ligado ao **grau de cooperação e comunicação com o usuário.**

AGILE MANIFESTO

Agile Manifesto

...um pouco de história

Em 2001 – um grupo de renomados desenvolvedores de software assinaram um documento denominado "**Manifesto para o desenvolvimento ágil de software**" ou

"AGILE MANIFESTO"

A lista dos **17 autores do Manifesto Ágil** reunidos em fevereiro de 2001, em Utah (EUA):

Kent Beck, Robert C. Martin, Alistair Cockburn, Ward Cunningham, Ron Jeffries, Stephen Mellor, Mike Beedle, Arie van Bennekum, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Brian Marick, Ken Schwaber, Jeff Shuterland, Dave Thomas, Jon Kern

Agile Manifesto

Indivíduos e interações

Mais que processos e ferramentas

Software funcionando

Mais que documentação abrangente

Colaboração com o cliente

Mais que negociação de contratos

Responder a mudanças

Mais que seguir um plano

Agile Manifesto

Princípios

1. **Nossa maior prioridade é satisfazer ao cliente com entrega contínua e adiantada de software com valor agregado.**
2. Mudanças de requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Os processos ágeis tiram vantagem das mudanças visando à vantagem competitiva para o cliente.
3. **Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.**
4. Pessoa de negócios e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.

Agile Manifesto

Princípios

5. **Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessários e confie neles para realizar o trabalho.**
6. O método mais eficiente e eficaz de transmitir informação para a equipe e entre a equipe de desenvolvimento é a conversa frente a frente.
6. **Software funcional é a medida primária do progresso.**
7. Processos ágeis promovem um desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante sempre.

Agile Manifesto

Princípios

9. **Contínua atenção à excelência técnica e bom projeto aumenta a agilidade.**
10. Simplicidade – a arte de maximizar a quantidade de trabalho não realizado – é essencial.
11. **As melhores arquiteturas, requisitos e projetos emergem de times auto-organizáveis.**
12. Em intervalos regulares, o time reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Agile Manifesto

O Movimento Ágil combina

Um pouco de filosofia
e
um conjunto de princípios de desenvolvimento de software.

E sugere:

- equipes de projetos pequenas;
- equipes altamente motivadas;
- métodos informais;
- artefatos mínimos de engenharia;
- simplicidade geral no desenvolvimento.

Agile Manifesto

AGILE manifesto – compreendendo princípios e valores

- É **uma atitude**, não refere-se a um processo prescritivo
- É **um suplemento aos métodos existentes**, não se apresenta como um modelo completo
- É **uma forma efetiva de se trabalhar em conjunto** para atingir as necessidades das partes interessadas no Projeto
- **Não** é um **ataque aos artefatos**, mas aconselha a criação de artefatos que tem valor
- **Não** é um **ataque às ferramentas CASE**

Agile Manifesto

Agile Manifesto e os Fatores humanos

Os **Fatores humanos** são enfatizados, pois são capazes de moldar o projeto e direcionar suas características.

- Multidisciplinaridade
- Auto-organização
- Foco comum
- Colaboração
- Habilidade na tomada de decisão

Agile Manifesto

Agile Manifesto e as Práticas comuns

- **Lista priorizada** de requisitos de negócios
- **Equipe coesa**, dedicada e multidisciplinar
- Trabalhar primeiramente nos **requisitos de maior prioridade** (valor de negócio)
- Entregar pequenas versões em **iterações curtas** (2 a 4 semanas)
- Demonstrar o **incremento de produto** ao final da iteração
- **Sincronizar as atividades** diariamente através de uma **reunião breve** e **objetiva**
- **Fomentar *feedback*** e **licões aprendidas** ao final de uma iteração
- **Atualizar e repriorizar a lista de requisitos** mediante o *feedback* e aprendizado recente

SCRUM

SCRUM

Objetivo

fornecer um processo conveniente para projetos e desenvolvimento de software

SCRUM PHASES

Planning Phase – fluxo linear com entradas e saídas definidas

Sprint – fluxo ciclíco (incremental) segue abordagem PDCA

Closure Phase – fluxo linear com entradas e saídas definidas

SCRUM

SCRUM – princípios

- **Equipes pequenas e multidisciplinares** trabalham de forma integrada e consideram arquitetos, desenvolvedores, engenheiros e testers
- **Equipes se auto-organizam** para planejar e desenvolver o trabalho de cada *Sprint*
- Trabalho em equipe facilitado pelo ***Scrum Master*** que deve **remover impedimentos** e **reforçar pontos centrais** do *SCRUM*
- Equipes trabalham a partir do ***Backlog do Produto*** que é **constantemente revisado e priorizado**
- **Reuniões de status diária** ocorrem ao longo de cada *Sprint*, devem durar cerca de 15 minutos e são denominadas – **Standup Meeting**

SCRUM

SCRUM – processos, artefatos e atores

- **Processos**
  - *Game, PreGame, PostGame, Sprints, Sprint Planning Meeting, Sprint Retrospective, Daily Meeting*
- **Artefatos**
  - *Product Baklog, Sprint Backlog, Post-Its, BurnDown*
- **Atores**
  - *Product Owner, Scrum Master, Scrum Team*

SCRUM – papéis

- Dono do produto
- Visão de retorno do projeto
- Prioriza requisitos
- Planeja releases
- Representante do cliente

- Líder do processo
- Manter o foco no processo
- Remover impedimentos
- Auxiliar na comunicação

- Pessoas que implementam o projeto
- Multidisciplinar
- Praticam autogestão
- Planeja sprint

SCRUM

SCRUM – fases

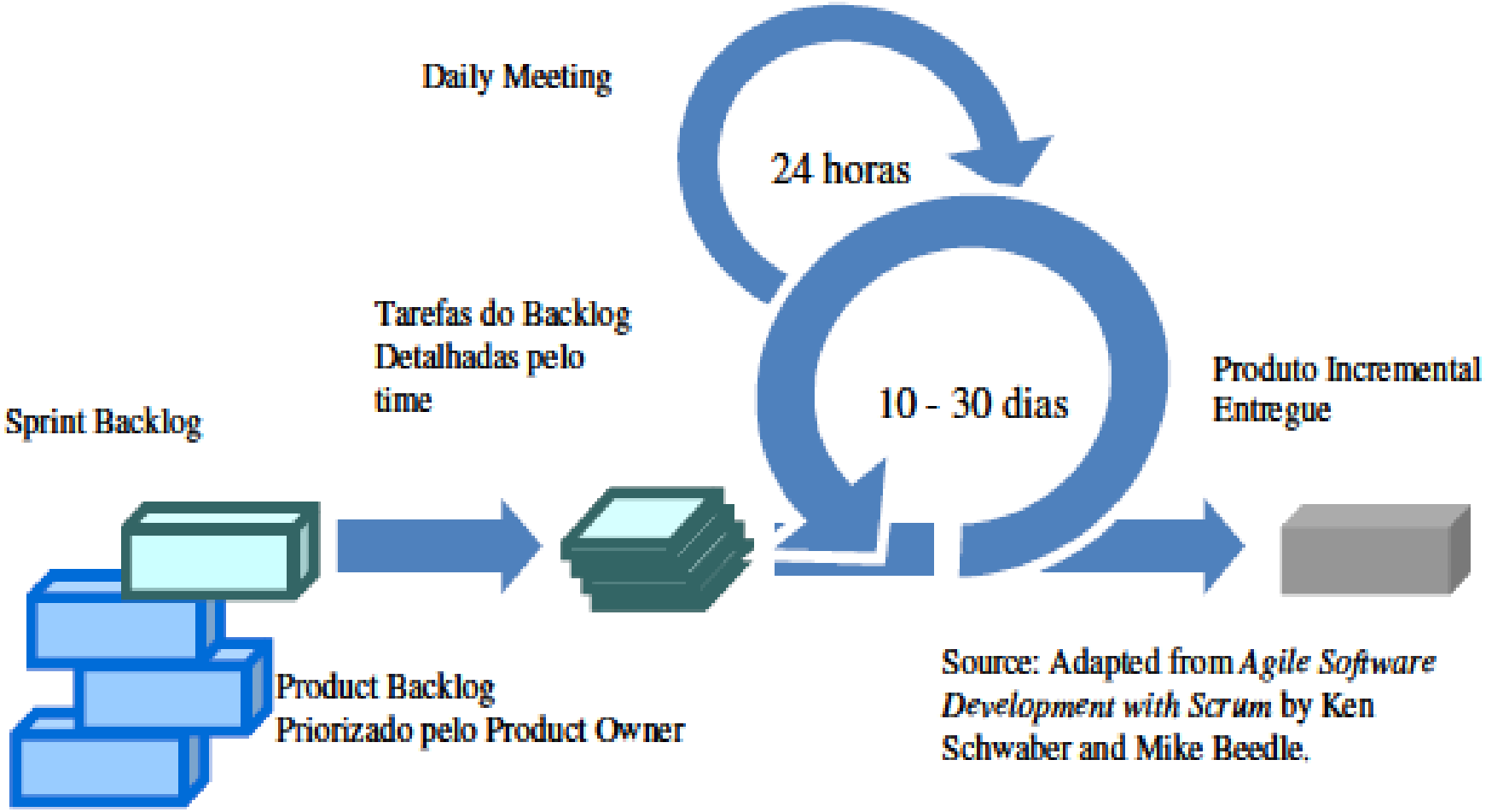
- ***Planning Phase* (Planejamento)**
  - Conhecimento de um *backlog* (requisitos) a serem desenvolvidos
  - Definição de datas de entrega
  - Priorização dos *releases*
  - Gestão de riscos
  - Estimativas de custo
- ***Architecture/High Level design***
  - Identificar mudanças necessárias para implementar um item de *backlog*
  - Refinar arquitetura para suportar o novo contexto
  - Identificar problemas ou questões para o desenvolvimento das mudanças

SCRUM

SCRUM – fases

- ***Sprint*** (Desenvolvimento)
  - Desenvolvimento de forma iterativa e incremental
  - O *sprint* não tem um tempo definido (um mês seria o padrão). Deve ser ajustado conforme feedbacks do dono do produto
  - Uma ***Sprint*** considera
    - Reunião de Planejamento de Sprint
    - Scrum diária (desenvolvimento da Sprint + *daily meeting*)
      - Implementação
      - *Wrap*: concluir uma versão que implementa os itens do *product backlog* (requisitos)
    - Revisão da Sprint - reuniões para rever o trabalho realizado
    - Retrospectiva da Sprint

SCRUM



Source: Adapted from *Agile Software Development with Scrum* by Ken Schwaber and Mike Beedle.

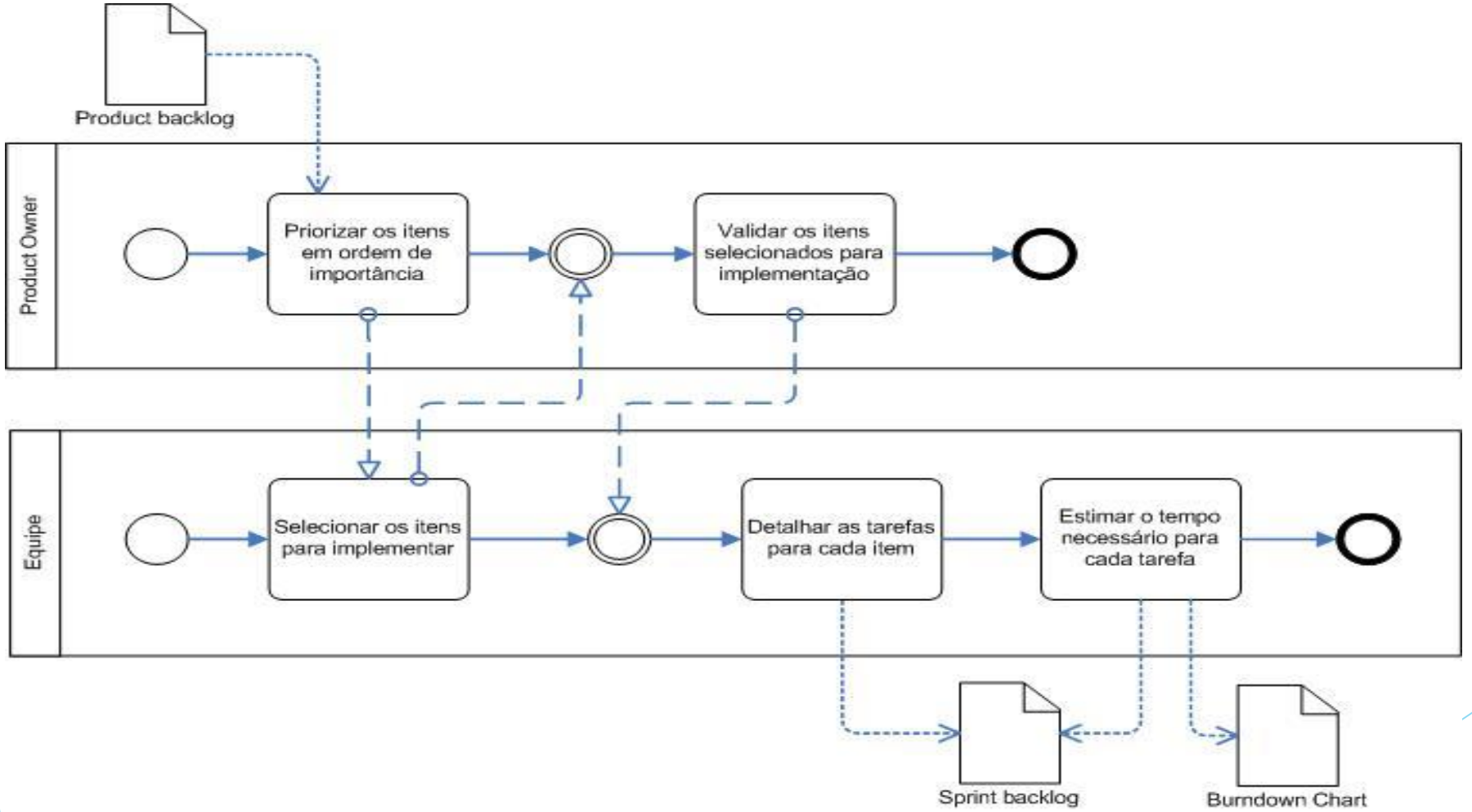
SCRUM

SCRUM – fases

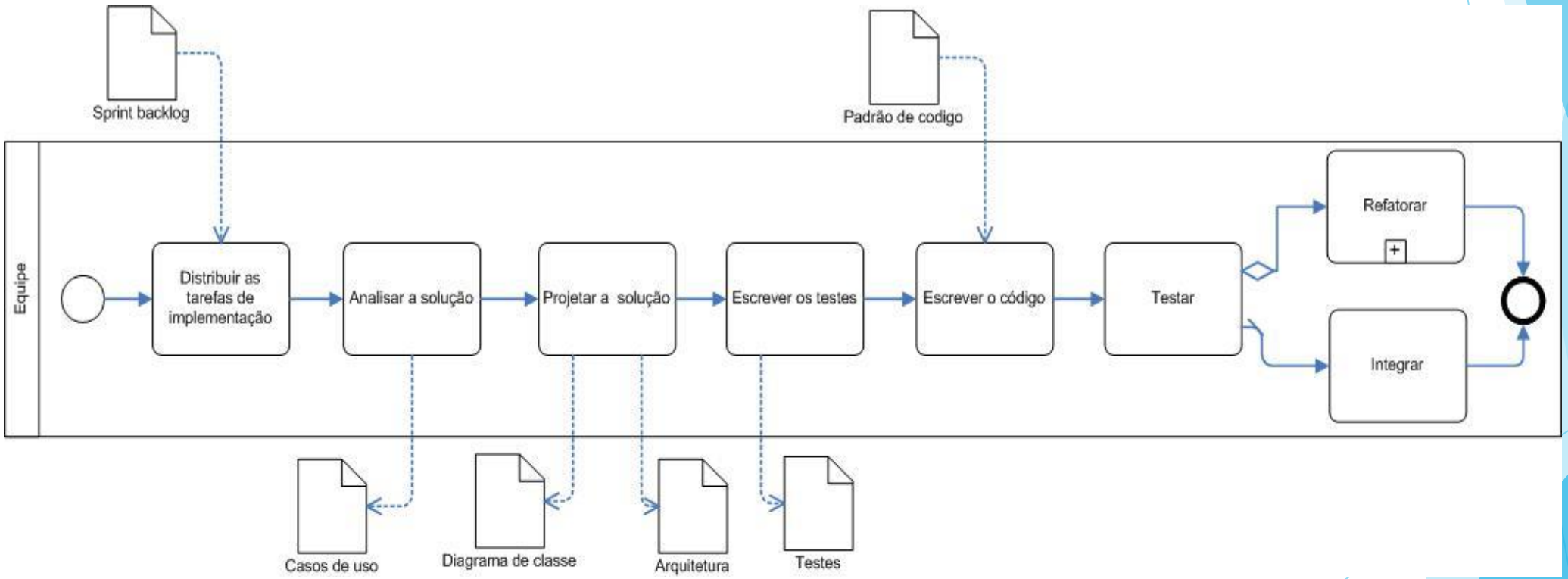
- ***Closure*** (Encerramento)
  - Prepara o produto desenvolvido para liberação
  - Envolve tarefas como treinamento, testes gerais e marketing

SCRUM

SCRUM – Planejamento do Sprint

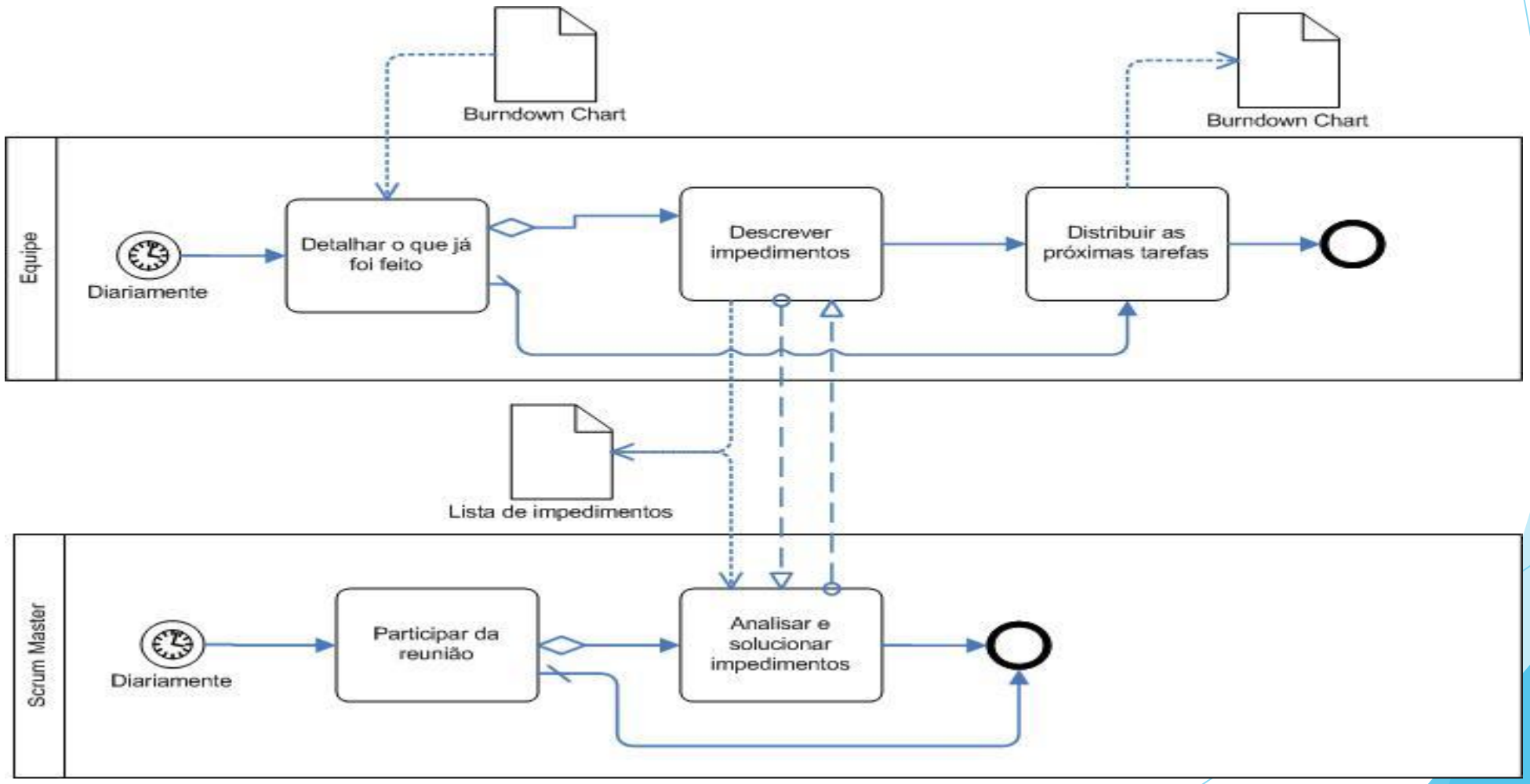


SCRUM – Desenvolvimento do Sprint



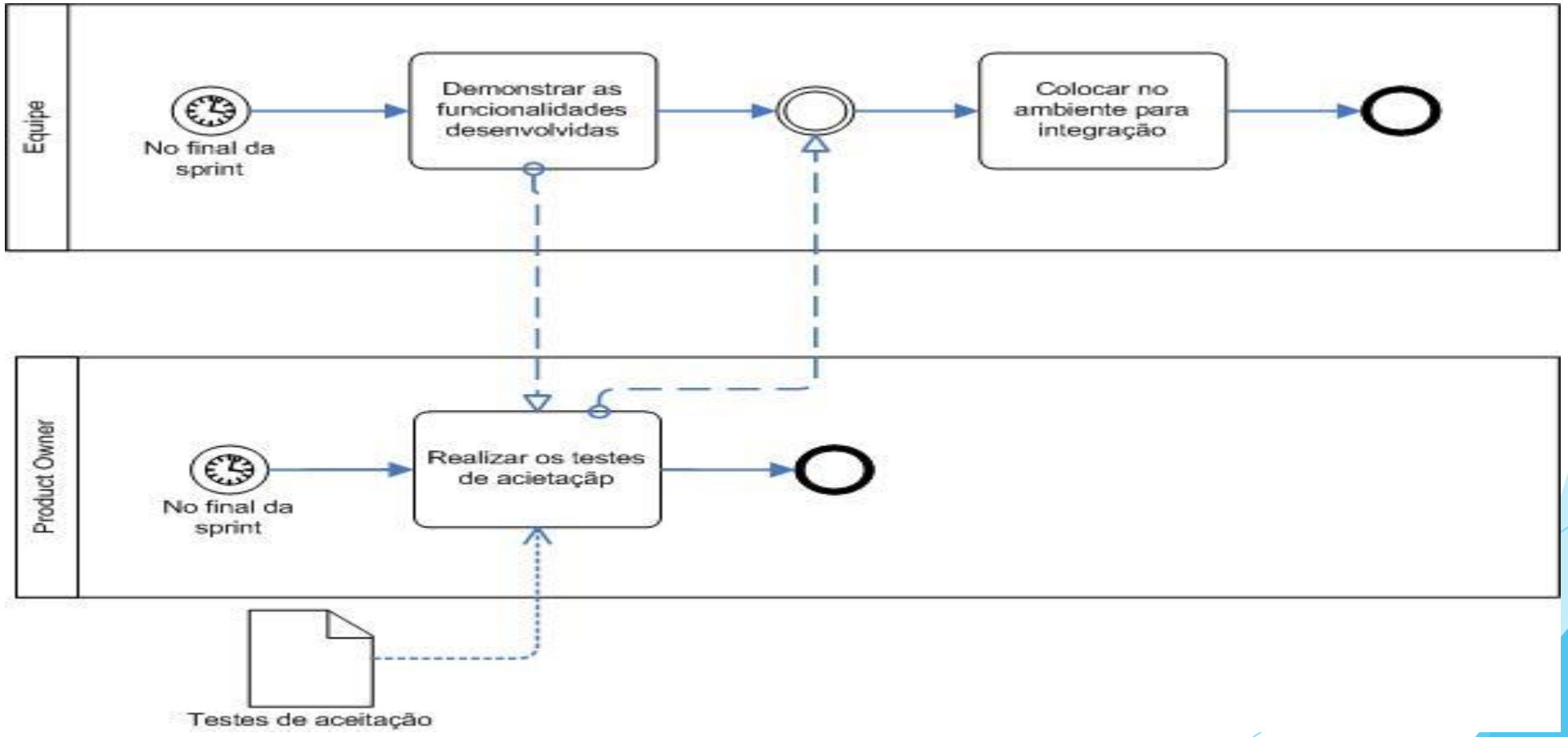
SCRUM

SCRUM – Reunião diária (Daily Meeting)



SCRUM

SCRUM – Revisão do Sprint (Sprint Review)



XP eXtreme Programming

XP eXtreme Programming

Extreme Programming: A gentle introduction

The first Extreme Programming project was started March 6, 1996. Extreme Programming is one of several popular Agile Processes. It has already been proven to be very successful at many companies of all different sizes and industries world wide.

Extreme Programming is successful because it stresses customer satisfaction. Instead of delivering everything you could possibly want on some date far in the future this process delivers the software you need as you need it. Extreme Programming empowers your developers to confidently respond to changing customer requirements, even late in the life cycle.

Extreme Programming emphasizes teamwork. Managers, customers, and developers are all equal partners in a collaborative team. Extreme Programming implements a simple, yet effective environment enabling teams to become highly productive. The team self-organizes around the problem to solve it as efficiently as possible.

make no sense, but when combined together a complete picture can be seen. The rules may seem awkward and perhaps even naive at first, but are based on sound values and principles.

Our rules set expectations between team members but are not the end goal themselves. You will come to realize these rules define an environment that promotes

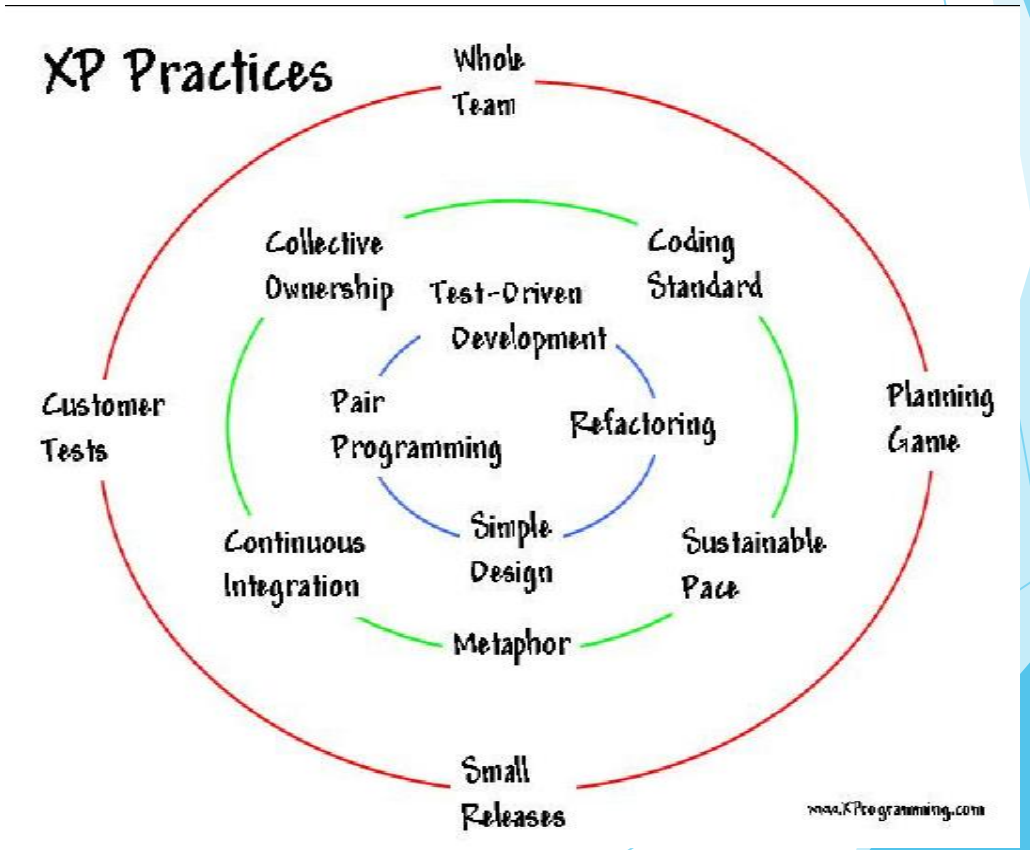
XP eXtreme Programming

XP (eXtreme Programming)

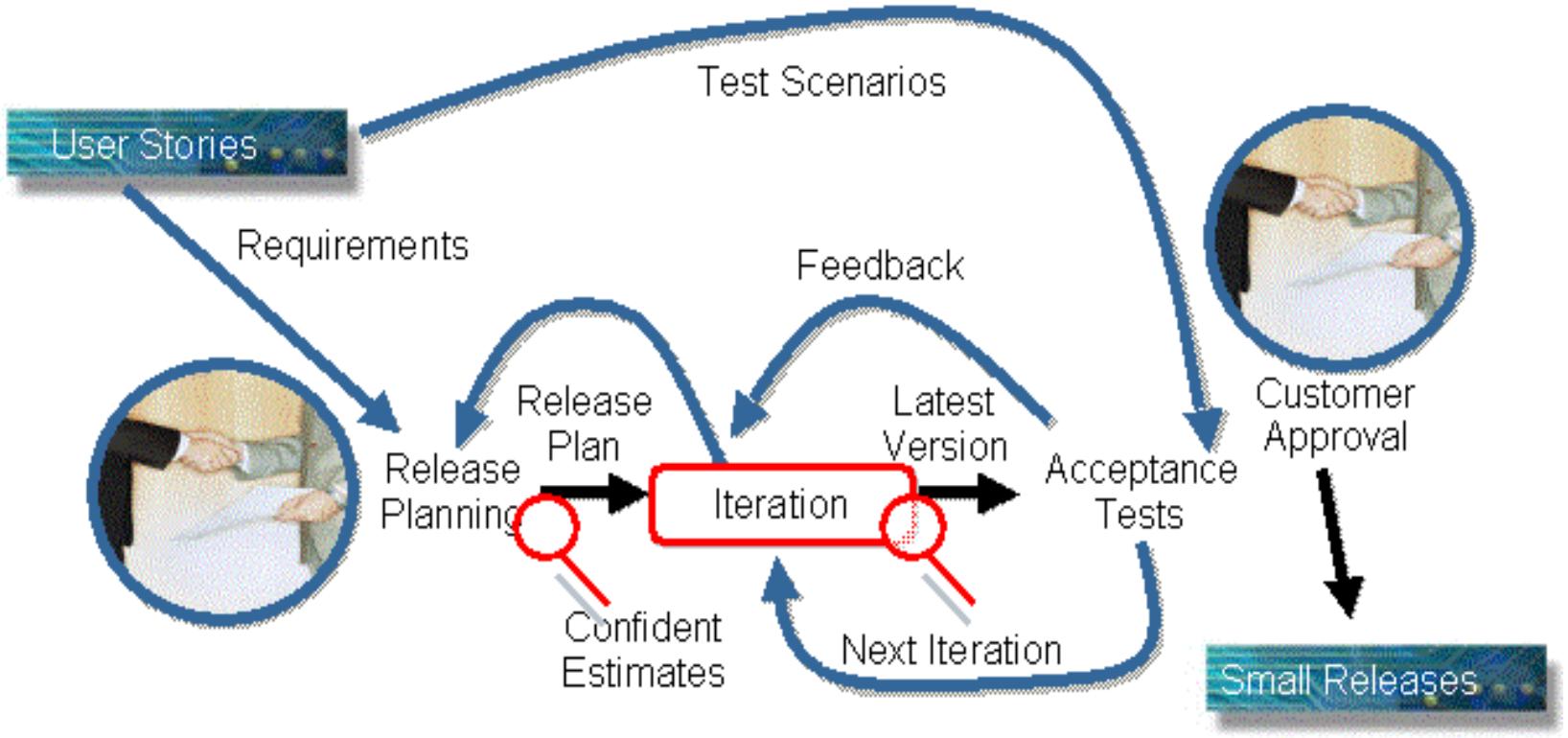
Projeto crítico na Chrysler para folha de pagamento – Kent Beck, Ward Cunningham e Ron Jeffries (1996)

Valores:

- **Comunicação**
- **Simplicidade**
- **Feedback**
- **Coragem**
- **Respeito**



XP eXtreme Programming



- Disciplina para desenvolvimento: entendimento e transparência entre desenvolvedores
- Arquitetura simples, bem como modelagem e estilo de codificação
- Coragem e aceitar que não sabe tudo. Desenvolvimento de software é uma atividade complexa
- Feedback constante para atender as necessidades do cliente
- Respeito para potencializar comunicação e feedback
- Principal tarefa é a codificação

Papéis:

Coach

Equipe

Tracker

Cliente

XP eXtreme Programming

The Rules of Extreme Programming

Planning

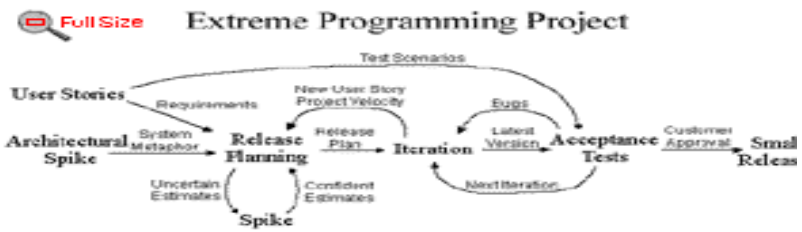
- User stories are written.
- Release planning creates the release schedule.
- Make frequent small releases.
- The project is divided into iterations.
- Iteration planning starts each iteration.

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.



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

XP eXtreme Programming

XP – práticas para o código

1. **Testes automatizados** – casos de testes são criados em paralelo ao desenvolvimento, por desenvolvedores e pelo cliente

2. **Refatorações** – melhoria técnica no código tornando-o mais legível, simples, organizado e preparado para acomodar novas funções

3. **Programação pareada** – cada par torna-se responsável por analisar, projetar, implementar, testar e integrar a funcionalidade

4. **Padronização de código** – acordos feitos entre os membros da equipe sobre como o código será escrito

XP eXtreme Programming

XP – práticas para o código

5. **Propriedade coletiva** – a versão do código no repositório é de responsabilidade de toda a equipe, não existe exclusividade

6. **Repositório de código** – agrupamento da última versão estável do código

7. **Integração contínua** – código fonte devem ser enviados frequentemente ao repositório para validação e disponibilização

8. **Buid ágil** – construção e automatização dos testes para colocar o software em funcionamento deve ser feita rapidademente

XP eXtreme Programming

XP eXtreme Programming

SCRUM e XP

- **Semelhança de princípios**
  - Equipes pequenas
  - Requisitos instáveis ou desconhecidos
  - Iterações curtas para prover visibilidade ao desenvolvimento
- **Dimensões diferenciadas**
  - *Scrum* divide o desenvolvimento em *sprints* de 30 dias e reuniões diárias de 15 minutos
  - As equipes são formadas por pessoas com competências diferentes: projetistas, programadores, engenheiros e gerentes de qualidade
  - *Scrum* possui um mecanismo de informação de status atualizado continuamente e a divisão de tarefas é explícita
- **SCRUM e XP** podem ser complementares, pois o **SCRUM** enfatiza práticas de gerenciamento e o **XP** enfatiza práticas integradas de engenharia do produto

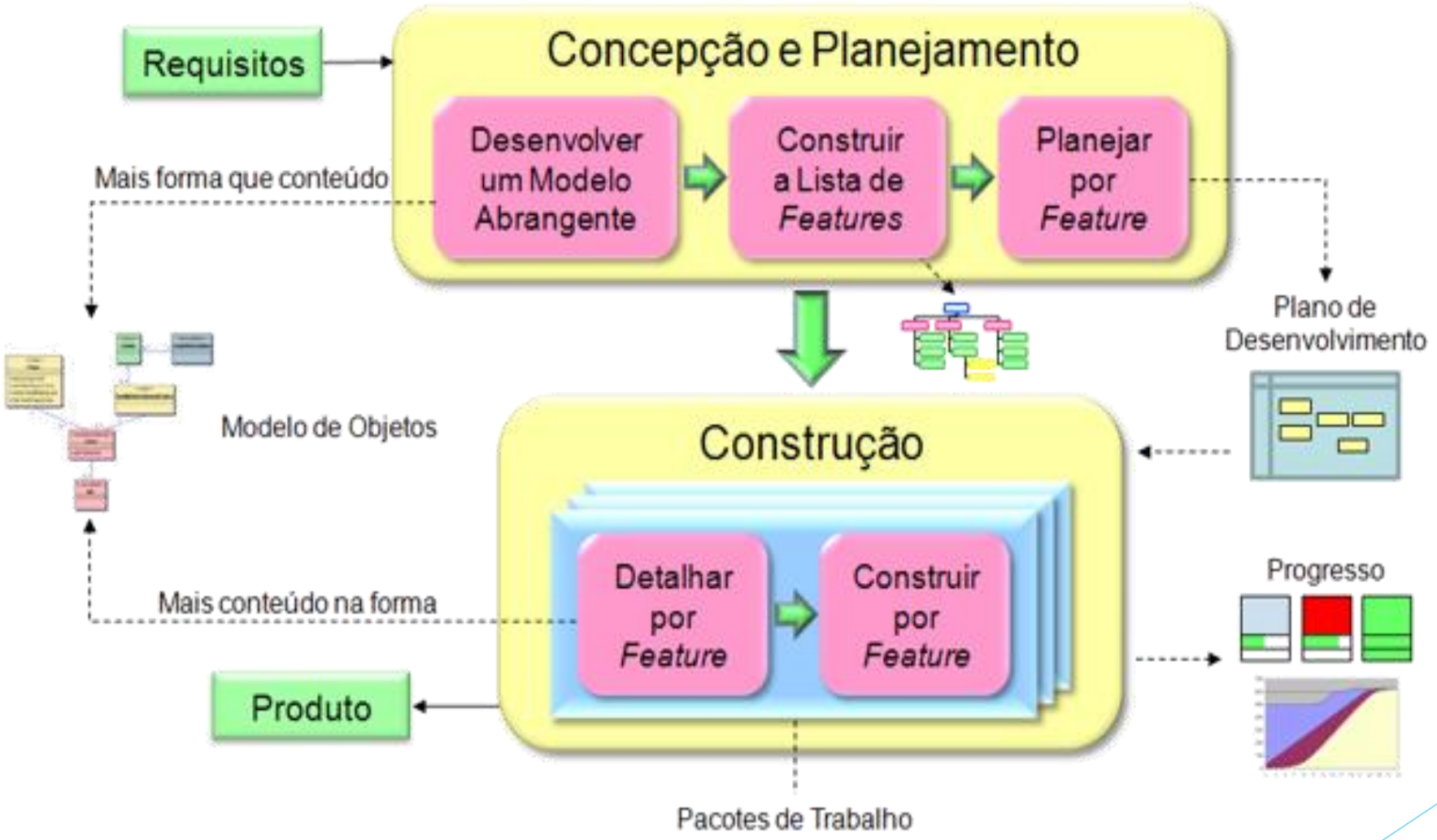
FDD

FEATURE DRIVEN DEVELOPMENT

FDD (Feature Driven Development)

- Desenvolvimento orientado a funcionalidades
- Origem: projeto em 1997-1999, em um banco de Cingapura (UOB (*United Overseas Bank*))
  - Situação:
    - 2 anos de consultoria,
    - 3500 páginas de casos de uso,
    - modelo com centenas de classes
    - projeto foi considerado inviável
  - Gestão de **Jeff de Luca** (um dos membros da equipe)
    - Convenceu a diretoria do banco
    - Pequena equipe de talentos de classe mundial
    - 15 meses depois, 2000 features entregues por equipe de 50 pessoas

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
Progresso
Mais conteúdo na forma
Produto
Pacotes de Trabalho

FDD

FDD (Feature Driven Development)

Funcionalidade (*feature*)

- Para o cliente: valor claro no contexto de seu domínio de negócio
- Para a equipe: não deve ocupar mais que uma iteração para ser desenvolvida, tipicamente menos que 2 semanas (80 horas)
- Para o tester: escrever casos de testes que comprovem o seu funcionamento

FDD

FDD (Feature Driven Development)

Cinco processos principais em suas fases lineares e iterativas

Concepção e Planejamento (Inicialização) (linear)

- DMA Desenvolver um Modelo Abrangente
- CLF Construir a Lista de Funcionalidades
- PPF Planejar Por Funcionalidade

Construção (Iterativa)

- DPF Detalhar Por Funcionalidade
- CPF Construir Por Funcionalidade

FDD

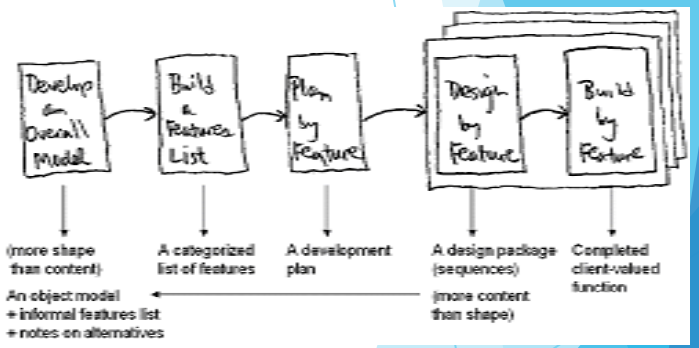
FDD (Feature Driven Development)

DMA (~Arquitetura)

- Definição do domínio do sistema, contexto e requisitos por meio de uma especificação das funcionalidades
- O domínio do projeto é dividido em domínios menores e mais específicos

CLF (*features*)

- Construção de uma lista de funcionalidades
- Cada conjunto de funcionalidades é uma função para uma área de domínio
- A lista é revisada pelos usuários e patrocinadores do sistema para sua aprovação



PPF

- Criação de um plano de alto nível, no qual as funções são sequenciadas de acordo com a prioridade e a dependência de cada uma, e então designadas ao time de programadores

FDD

FDD (Feature Driven Development)

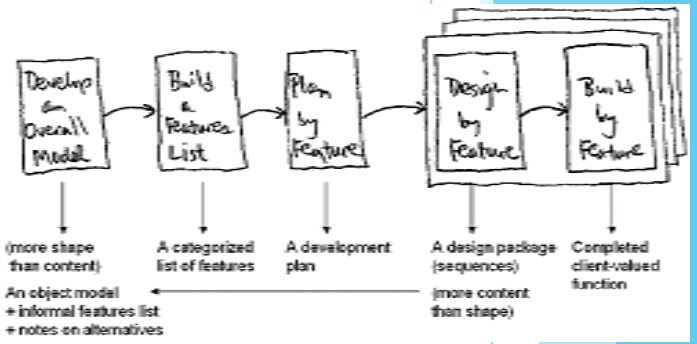
A cada iteração:

DPF

- Consiste em preparar um pacote para ser completamente implementado pelos desenvolvedores

CPF

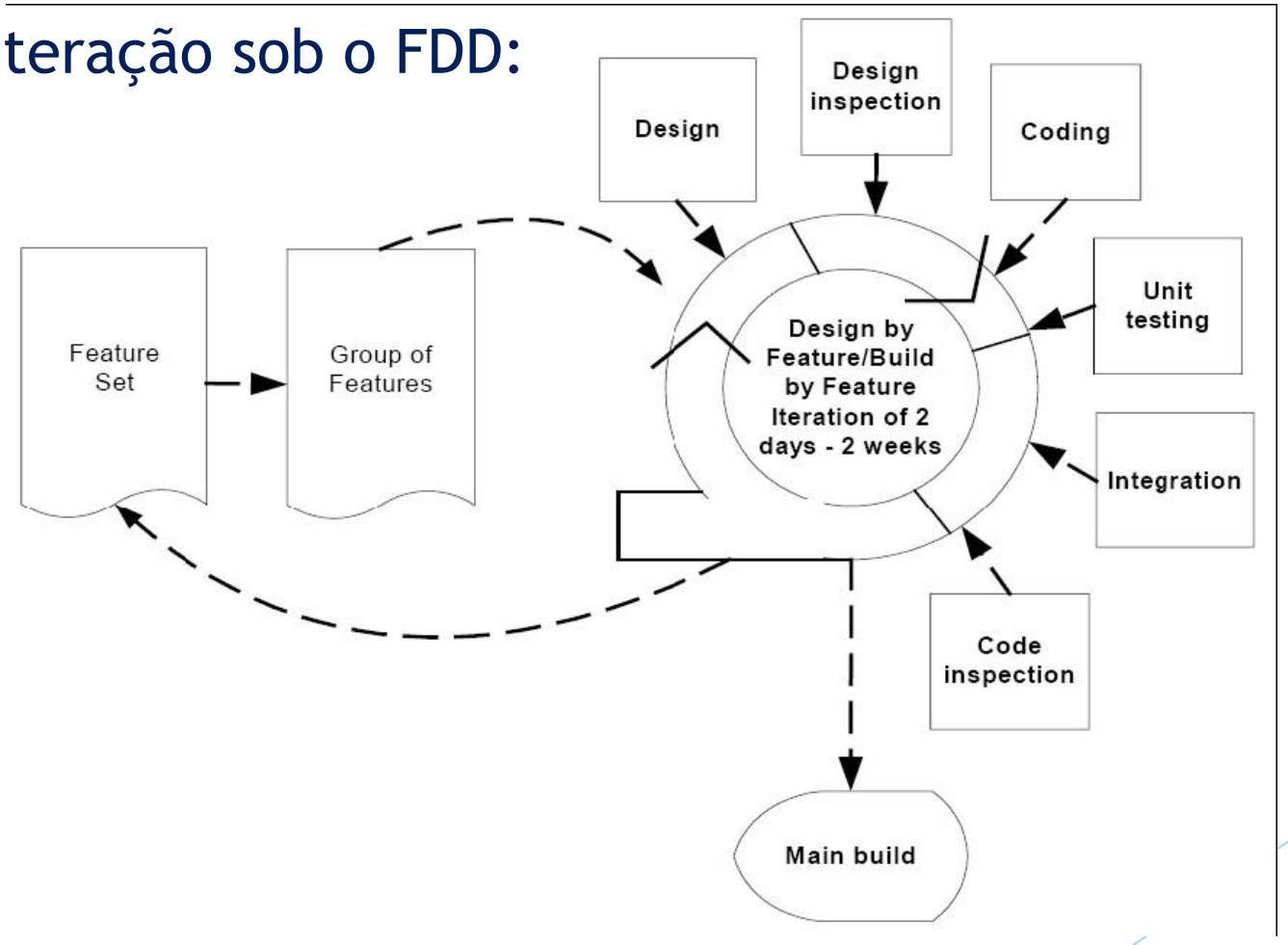
- Consiste na parte iterativa do método. Um pequeno grupo de funcionalidade é selecionado, assim como as funções necessárias para o desenvolvimento desse pacote.



FDD

FDD (Feature Driven Development)

Uma iteração sob o FDD:



FDD

FDD (Feature Driven Development)

- Equipe
  - Especialistas de domínio
  - Gerentes de Projeto e de Desenvolvimento
  - Programadores líderes e Proprietários de Classes
  - Arquiteto Líder

FDD (Feature Driven Development)

- **Modelagem de objetos de domínio**
  - Exploração e explicação do problema do domínio
  - Resulta em um framework
- **Desenvolvimento por funcionalidade**
  - Desenvolvimento e acompanhamento do progresso através de da lista de funcionalidades
- **Proprietários de classes individuais/código**
  - Cada classe possui um único desenvolvedor responsável
- **Equipes de funcionalidades**
  - Formação de equipes pequenas e dinâmicas
  - Inspeção (*Inspection*)
  - Uso dos melhores métodos conhecidos de detecção de erros

FDD (Feature Driven Development)

- **Inspeções**
  - No DPF ocorrem as inspeções de desenho das funcionalidades
  - No CPF ocorrem as inspeções de código-fonte relativos as funcionalidades
- **Montagens (*builds*) frequentes**
  - Em intervalos regulares, compilar o sistema para antecipar erros de integração tornando-o sempre disponível e demonstrável
  - Capacita os testes de integração e regressão
  - Proporciona condição de auditorias e medições
- **Gestão de Configuração**
  - Habilita acompanhamento do histórico do código-fonte
- **Relatório e visibilidade de resultados**
  - Proporcionar visão acurada da situação atual do projeto e saber o quão rápido a equipe adiciona funcionalidade

FDD

FDD (Feature Driven Development)

- Características principais (resumo)
  - Método ágil e adaptativo
  - Foco nas fases de desenho e construção
  - Interage com outras metodologias
  - Não exige nenhum processo específico de modelagem
  - Possui desenvolvimento iterativo
  - Enfatiza aspectos de qualidade durante o processo e inclui entregas frequentes e tangíveis
  - Suporta desenvolvimento ágil com rápidas adaptações às mudanças de requisitos e necessidades do mercado

LD
LEAN SOFTWARE DEVELOPMENT

LD

LD Lean Software Development

- Idealizado por Mary Poppendieck (2000)
- Focado na identificação de gargalos no processo de desenvolvimento de software
- Metáfora de chão de fábrica

- Inspirações:
  - ***TQM Total Quality Management*** **(Deming, anos 50)**
  - ***Lean Production*** **– Toyota (Japão, anos 50)**
  - **Teoria de Sistemas Dinâmicos (MIT, anos 60)**
  - ***Lean Construction*** **(adaptabilidade na construção civil, anos 90)**

LD Lean Software Development

- Realizar mais **o que importa,** eliminando **o que não importa**
- Trabalha com a idéia que **soluções elegantes** quer devem ser criadas com **simplicidade**
- Baseia-se nos princípios do **TPS - *Toyota Production System*** – também chamado de Produção Enxuta e **Lean Manufacturing**
  - O modelo tem o objetivo de aumentar a eficiência da produção pela eliminação contínua de desperdícios

LD

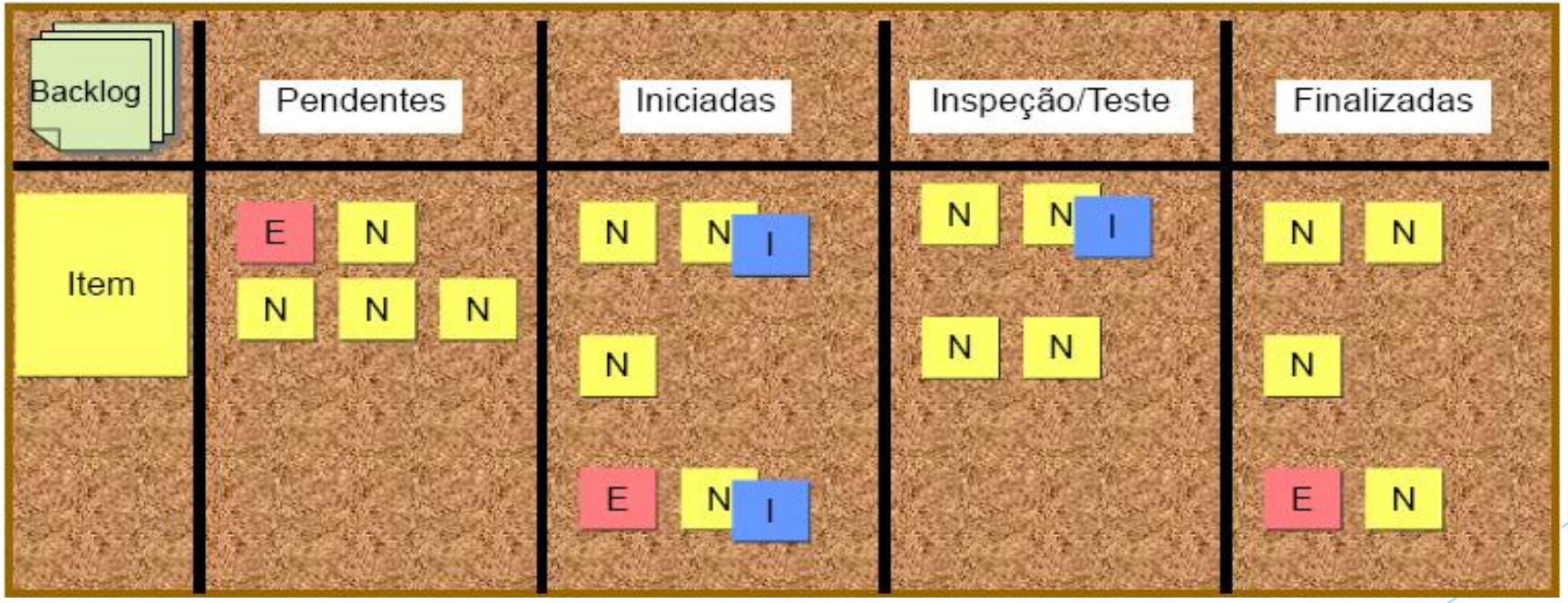
LD Lean Software Development

Práticas ***TPS - Toyota Production System (Lean Manufacturing)***

- ***Kanban*** - placa visível para comunicar status dentro um processo
- ***Heijunka*** - nivelamento da produção
- ***Kaizen*** - "kai" significa "mudança" ou "ato de correção" e "zen“ significa "bom" – ou seja, melhoria contínua
- ***Poka-Yoke*** - é um dispositivo físico de controle de qualidade, que é acionado automaticamente quando há algum erro ou defeito no processo de produção
- ***Hansei*** - reflexões profundas em busca da melhoria contínua

LD Lean Software Development

Exemplo de Comunicação com Kanban (Lean)



LD

LD Lean Software Development

O pensamento ***Lean*** em software

Fatos

- Mercado é mais rápido que especificação ou requisito
- Dificuldade de adaptação
- Controle rígido de escopo
- Tomada de decisão centralizada
- Qualidade sempre opcional

Desafios

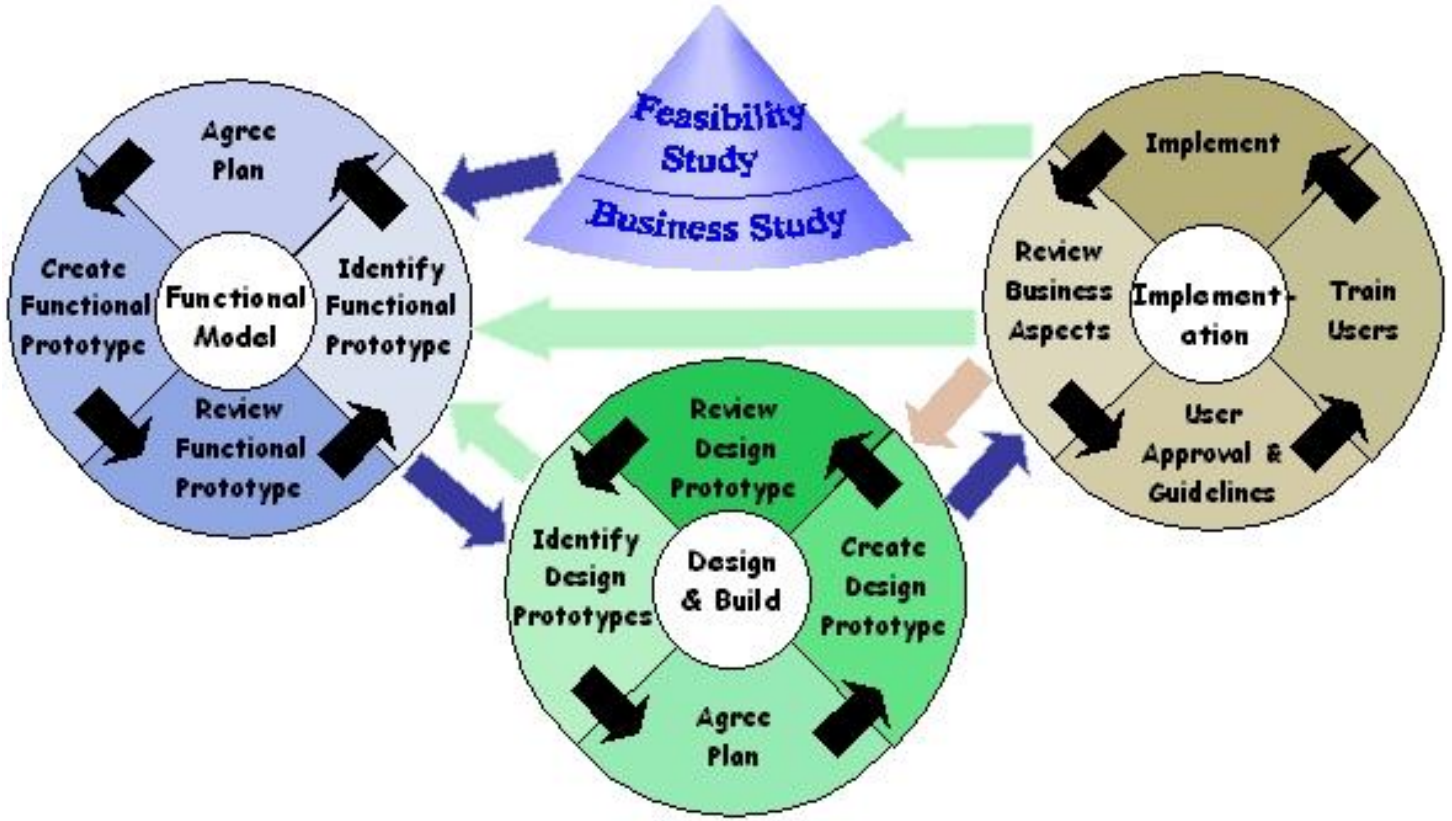
- Elimine desperdícios
- Construa com qualidade
- Entregue rápido
- Respeite as pessoas
- Crie conhecimento
- Adie comprometimentos
- Otimize o todo

Apêndices

DSDM

DYNAMIC SYSTEM DEVELOPMENT METHOD

DSDM (Dynamic System Development Method)



DSDM

DSDM (Dynamic System Development Method)

- Fases
  - ***Feasibility study*** (Estudo de viabilidade)
  - ***Business study*** (Estudo dos requisitos de negócio)
  - ***Functional model iteration*** (Iteração do modelo funcional)
    - Baseado na criação de um protótipo do produto
  - ***Design and build iteration*** (Iteração de projeto e construção)
  - ***Implementation iteration*** (Implementação final)

DSDM (Dynamic System Development Method)

- **Princípios:**
  - Iterações fixas (2-6 semanas)
  - Releases frequentes
  - Adaptabilidade a mudanças de requisitos

- **Ciclo de Vida:**
  - Estudo de viabilidade
  - Estudo do negócio
  - 3 ciclos em paralelo, entrelaçados
    - Ciclo do modelo funcional - análise e protótipos
    - Ciclo de design e build - engenharia do produto
    - Ciclo de implementação - implantação operacional

DSDM

DSDM (Dynamic System Development Method)

- **Papéis e Responsabilidades**
  - Desenvolvedores (*Developers*)
  - Desenvolvedores Sêniores (*Senior Developers*)
  - Coordenador Técnico (*Technical Coordinator*)
  - Usuário Embaixador (*Ambassador User*)
  - Usuário Consultor (*Adviser User*)
  - Visionário (*Visionary*)
  - Executivo responsável (*Executive Sponsor*)
  - Especialista do domínio (*Domain experts*)
  - Gerente do domínio (*Domain manager*)

DSDM

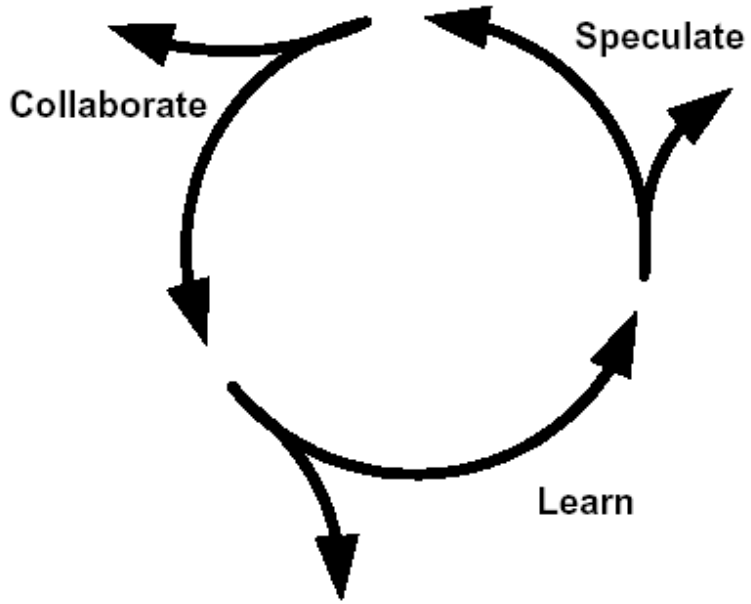
- Método inspirado no XP
- *Framework* para desenvolvimento rápido de aplicações (RAD)
- Fixa o tempo e os recursos ajustando a quantidade de funcionalidades a desenvolver
- Aplicável em pequenas equipes
- Suporta mudanças nos requisitos durante o ciclo de vida de desenvolvimento
- Usuário sempre envolvido
- Equipe autorizada a tomar decisões

- Foco na entrega freqüente de produtos
- Adaptação ao negócio é o critério para entregas
- “Construa o produto certo antes de você construí-lo corretamente”
- O desenvolvimento é iterativo e incremental
- Mudanças são reversíveis utilizando pequenas iterações
- Requisitos são acompanhados em alto nível
- Testes integrados ao ciclo de vida

ASD
ADAPTIVE SOFTWARE DEVELOPMENT

ASD Adaptive Software Development

Ciclos de 3 fases



- ***Speculate* (Especular)**
  - Fixam-se prazos e objetivos
  - Definição de um plano baseado em componentes
- ***Collaborate* (Colaborar)**
  - Construção concorrente de vários componentes
- ***Learn* (Aprender)**
  - Repetitivas revisões de qualidade e foco na demostração das funcionalidades desenvolvidas (Learning loop)
  - Presença do cliente e especialistas do domínio

ASD

ASD Adaptive Software Development

- Ciclo:
- **→ Colaboração → Especulação→Aprendizado**

- Abordagem:
  - ***Do it wrong the first time***: erre cedo, corrija cedo, não potencialize mal-entendidos
  - ***Good enough quality***: melhor compromisso entre dimensões de qualidade (extrínseca e intrínseca) para os recursos disponíveis
  - Mecânica: RAD (***Rapid Application Development***), sessões JAD (***Joint Application Development***) com o cliente

ASD

ASD Adaptive Software Development

- **Características principais**
- **Iterativo e incremental**
  - Desenvolvimento em ambientes definidos e compreendidos
  - Preferível refazer do que tentar fazer corretamente já na primeira vez
  - Os ciclos duram de 4 a 8 semanas
- **Voltado para sistemas grandes e complexos**
- **Cliente sempre presente**
  - Desenvolvimento de aplicações em conjunto (*Joint Application development* – JAD).
- **Orientado a missões**
  - Atividades são justificadas através de uma missão, que pode mudar ao longo do projeto

ASD Adaptive Software Development

- **Características principais (cont.)**
- **Baseado em componentes**
  - Construir o sistema em pequenas partes
- **Prazos pré-fixados**
- **Tolerância a mudanças (*Change-tolerant*)**
  - As mudanças são frequentes
  - É sempre melhor estar pronto a adaptá-las do que controlá-las
  - Constante avaliação de quais componentes podem mudar
- **Orientado a riscos (*Risk driven*)**
  - Itens de alto risco são desenvolvidos primeiro

CRYSTAL CLEAR

Crystal

Crystal / Clear – metodologia direcionada a projetos pequenos, com equipes de até 6 desenvolvedores

Crystal / Clear foi criada por Alistair Cockburn (IBM – anos 90)

- Premissas
  - Todo projeto tem necessidades, convenções e uma metodologia diferente
  - O funcionamento do projeto é influenciado por fatores humanos,e há melhora quando os indivíduos produzem melhor
  - Comunicação melhor e lançamentos frequentes reduzem a necessidade de construir produtos intermediários
  - Assim como o SCRUM, os membros da equipe tem especialidades distintas
  - Existe uma forte ênfase na comunicação entre os membros do grupo
- Existem outras metodologias da família Crystal para grupos maiores

Crystal

Crystal Clear

- Para cada projeto há
- quatro parâmetros que determinam o método de desenvolvimento:
  - Tamanho da equipe
  - Localização geográfica
  - Criticalidade/Segurança
  - Recursos

Crystal Clear

- Toda a especificação e projeto são feitos informalmente, utilizando quadros publicamente visíveis
- Os requisitos são elaborados utilizando casos de uso, de forma similar às estórias de usuário em XP, onde são enunciados os requisitos que são transformados como tarefas e um processo é escolhido para sua execução
- As entregas das novas versões de software são feitos em incrementos regulares de um mês, e existem alguns subprodutos do processo que são responsabilidade de membros específicos do projeto
- A metodologia é propositalmente pouco definida para permitir que cada organização implemente as atividades que lhes pareçam mais adequadas. O que se fornece é um mínimo de suporte útil à documentação e à comunicação

Referências

BOEHM, B. “**Get ready for Agile methods, with care**”, IEEE Computer, 2002.

COCKBURN, A. & HIGHSMITH, J. “**Agile Software Development: the people factor**”, IEEE Computer, 2001.

CHOUDHARY, B. & RAKESH, S. K. “**An approach using agile method for software development**”, In: Proceedings of International Conference on Innovation and Challenges in Cyber Security 2016 (ICICCS-INBUSH), 2016.

HIGHSMITH, J. **Agile Software Development Ecosystems**., Addison-Wesley, 2003.

HIGHSMITH, J. & COCKBURN, A. “**Agile Software Development: the business of innovation**”, IEEE Computer, 2001.

MELO, C. D. O.; SANTOS, V.; KATAYAMA, E.; CORBUCCI, H.; PRIKLADNICKI, R.; GOLDMAN, A. & KON, F. “**The evolution of agile software development in Brazil**”, Journal of the Brazilian Computer Society, 2013.

PRIES, K.H. & QUIGLEY, J.M. **Scrum Project Managment**. CRC Press. 2011.

PRIKLADNICKI, R.; WILLI, R. & MILANI, R., **Métodos ágeis para desenvolvimento de software**. Porto Alegre: Bookman, 2014.