



Desenvolvimento Ágil

Herysson R. Figueiredo
herysson.figueiredo@ufn.edu.br



Sumário

- O que é desenvolvimento ágil
- O que é agilidade
- Agilidade e o custo das mudanças
- Processo ágil
- Princípios da agilidade
- Modelos de processo ágeis



O que é?

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 projeto pequenas e altamente motivadas; métodos informais; artefato de engenharia de software mínimos e, acima de tudo, simplicidade no desenvolvimento geral.



O que é?

Os princípios de desenvolvimento priorizam a entrega mais que análise e projeto (embora essas atividades não sejam desencorajadas); também priorizam a comunicação ativa e contínua entre desenvolvedores e clientes.



Quem realiza?

Os engenheiros de software e outros envolvidos no projeto (gerentes, clientes, usuários finais) trabalham conjuntamente em uma equipe ágil — uma equipe que se auto-organiza e que controla seu próprio destino. Uma equipe ágil acelera a comunicação e a colaboração entre todos os participantes (que estão a seu serviço).



Por que é importante?

O moderno ambiente dos sistemas e dos produtos da área é acelerado e está em constante mudança. A engenharia de software ágil constitui uma razoável alternativa para a engenharia convencional voltada para certas classes de software e para certos tipos de projetos, e tem se mostrado capaz de entregar sistemas corretos rapidamente.



Quais são as etapas envolvidas?

O desenvolvimento ágil poderia ser mais bem denominado “engenharia de software flexível”. As atividades metodológicas básicas permanecem: comunicação, planejamento, modelagem, construção e emprego.



Quais são as etapas envolvidas?

Entretanto, estas se transformam em um conjunto de tarefas mínimas que impulsiona a equipe para o desenvolvimento e para a entrega (pode-se levantar a questão de que isso é feito em detrimento da análise do problema e do projeto de soluções).



Qual é o artefato?

Tanto o cliente como o engenheiro têm o mesmo parecer: o único artefato realmente importante consiste em um “incremento de software” operacional que seja entregue, adequadamente, na data combinada.



Como garantir que o trabalho foi realizado corretamente?

Se a equipe ágil concordar que o processo funciona e essa equipe produz incrementos de software passíveis de entrega e que satisfaçam o cliente, então, o trabalho está correto.



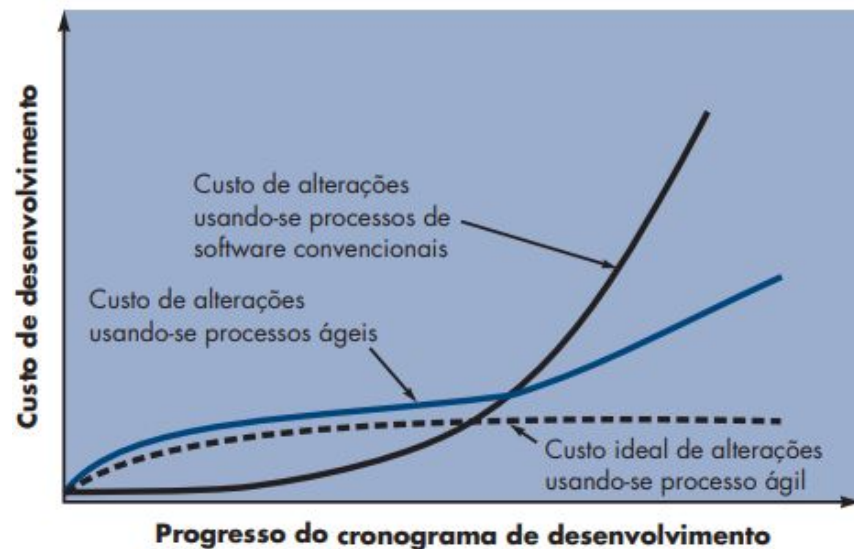
O que é Agilidade?

Uma equipe ágil é aquela rápida e capaz de responder apropriadamente a **mudanças**. **Mudanças** têm muito a ver com desenvolvimento de software. **Mudanças** no software que está sendo criado, **mudanças** nos membros da equipe, **mudanças** devido a novas tecnologias, **mudanças** de todos os tipos que poderão ter um impacto no produto que está em construção ou no projeto que cria o produto.

Os métodos ágeis são algumas vezes conhecidos como métodos *light* ou métodos enxutos (*lean methods*)

Agilidade e o Custo das Mudanças

A sabedoria convencional em desenvolvimento de software (baseada em décadas de experiência) afirma que os custos de mudanças aumentam de forma não linear conforme o projeto avança





O que é Processo Ágil?


Um **processo ágil** de software deve se **adaptar incrementalmente**. Para conseguir uma adaptação incremental, a equipe ágil precisa de *feedback* do cliente (de modo que as adaptações apropriadas possam ser feitas). Um efetivo catalisador para *feedback* de cliente é um **protótipo operacional** ou parte de um sistema operacional. Dessa forma, deve se instituir uma estratégia de **desenvolvimento incremental**. Os incrementos de *software* (**protótipos executáveis** ou **partes de um sistema operacional**) devem ser entregues em **curtos períodos de tempo**, de modo que as **adaptações** acompanhem o mesmo ritmo das mudanças (**imprevisibilidade**).



Princípios da agilidade

1- A maior prioridade é satisfazer o cliente por meio de entrega adiantada e contínua de software valioso.

2-Acolha bem os pedidos de alterações, mesmo atrasados no desenvolvimento. Os processos ágeis se aproveitam das mudanças como uma vantagem competitiva na relação com o cliente



Manifesto para Desenvolvimento Ágil de Software

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

Indivíduos e interações 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

Ou seja, mesmo havendo valor nos itens à direita,
valorizamos mais os itens à esquerda.

<https://agilemanifesto.org/iso/ptbr/manifesto.html>



Princípios da agilidade

3-Entregue software em funcionamento frequentemente, de algumas semanas para alguns meses, dando preferência a intervalos mais curtos.

4-O pessoal comercial e os desenvolvedores devem trabalhar em conjunto diariamente ao longo de todo o projeto.

5-Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e apoio necessários e confie neles para ter o trabalho feito



Princípios da agilidade

6-O método mais eficiente e efetivo de transmitir informações para e dentro de uma equipe de desenvolvimento é uma conversa aberta, de forma presencial.

7-Software em funcionamento é a principal medida de progresso

8-Os processos ágeis promovem desenvolvimento sustentável. Os proponentes, desenvolvedores e usuários devem estar capacitados para manter um ritmo constante indefinidamente



Princípios da agilidade

9-Atenção contínua para com a excelência técnica e para com bons projetos aumenta a agilidade.

10-Simplicidade — a arte de maximizar o volume de trabalho não efetuado — é essencial.

11-As melhores arquiteturas, requisitos e projetos emergem de equipes que se auto-organizam

12 - A intervalos regulares, a equipe se avalia para ver como tornar-se mais eficiente, então sintoniza e ajusta seu comportamento de acordo



Modelos de Processos Ágeis

Extreme Programming - XP

Scrum

Método de desenvolvimento de sistemas dinâmicos (*Dynamic Systems Development Method, DSDM*)

Desenvolvimento dirigido a Funcionalidades (*Feature Drive Development, FDD*)

Extreme Programming - XP



Extreme Programming - XP

O XP (*Extreme Programming*), ou Programação Extrema, foi criado por Kent Beck, um engenheiro de software americano. Ele desenvolveu a metodologia no final da década de 1990. O processo começou formalmente em 1996, quando Kent Beck assumiu a liderança de um projeto na Chrysler chamado Chrysler Comprehensive Compensation System (C3).

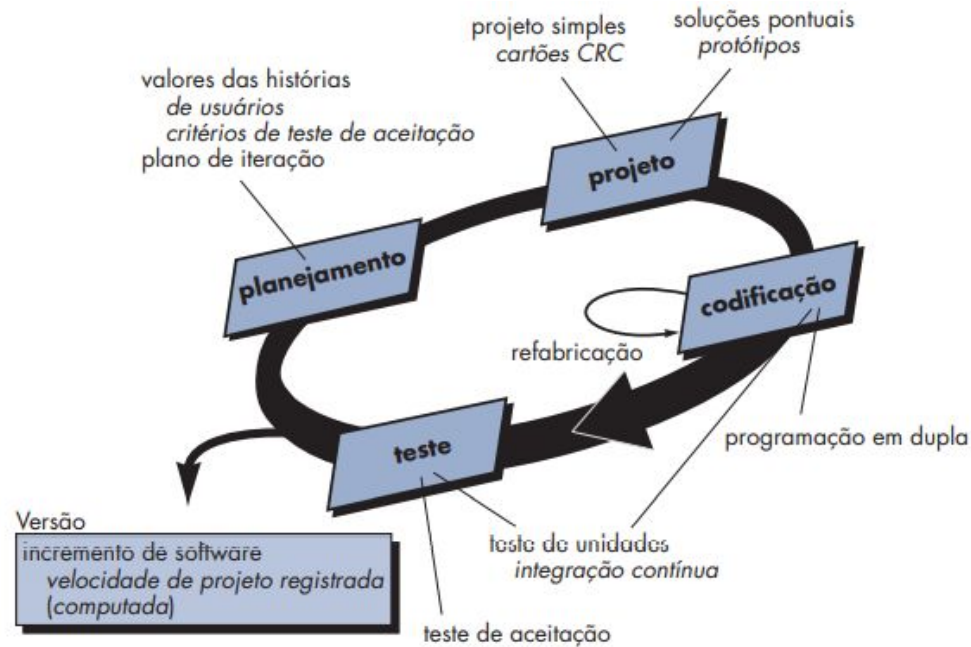
Foi nesse projeto que ele refinou e juntou um conjunto de boas práticas de desenvolvimento, que mais tarde foram formalizadas e popularizadas com a publicação do seu livro "Extreme Programming Explained: Embrace Change" em 1999.



Extreme Programming - XP

A *Extreme Programming* (programação extrema) emprega uma abordagem orientada a objetos como seu paradigma de desenvolvimento preferido e envolve um conjunto de regras e práticas constantes no contexto de quatro atividades metodológicas: planejamento, projeto, codificação e testes.

Extreme Programming - XP





Extreme Programming - XP

Planejamento : A atividade de planejamento (também denominada o jogo do planejamento) se inicia com a atividade de ouvir uma atividade de **levantamento de requisitos** que capacita os membros técnicos da equipe XP a entender o ambiente de negócios do software e possibilita que se consiga ter uma percepção ampla sobre os resultados solicitados, fatores principais e funcionalidade.



Extreme Programming - XP

A atividade de “Ouvir” conduz à criação de um conjunto de “**histórias**” (também denominado histórias de usuários) que descreve o resultado, as características e a funcionalidade requisitados para o software a ser construído.



Extreme Programming - XP

Projeto: O projeto XP segue rigorosamente o princípio **KIS** (*keep it simple*, ou seja, preserva a simplicidade). É preferível sempre um projeto simples do que uma representação mais complexa. Como acréscimo, o projeto oferece um guia de implementação para uma história à medida que é escrita nada mais, nada menos. O projeto de funcionalidade extra (pelo fato de o desenvolvedor supor que ela será necessária no futuro) é desencorajado.



Extreme Programming - XP

Projeto: Como o projeto XP não usa praticamente nenhuma notação e produz poucos, se algum, artefatos, além dos cartões CRC e soluções pontuais, o projeto é visto como algo transitório que pode e deve ser continuamente modificado conforme a construção prossegue.

Extreme Programming - XP

Ex: CRC

Class Responsibilities and Collaborators

| | | | |
|------------------------|--------------------------------|-------------------------|--------------------|
| Classe: Conta Corrente | | | Classes associadas |
| Responsabilidade | | Colaboração | |
| atributos | Saber o seu saldo | Cliente | |
| | Saber seu cliente | Histórico de Transações | |
| | Saber seu número | | |
| métodos | Manter histórico de transações | | |
| | Realizar saques e depósitos | | |



Extreme Programming - XP

Codificação: Depois de desenvolvidas as histórias e o trabalho preliminar de elaboração do projeto ter sido feito, a equipe não passa para a codificação, mas sim, desenvolve uma série de testes de unidades que exercitam cada uma das histórias a ser incluídas na versão corrente (incremento de software).



Extreme Programming - XP

Codificação: Uma vez criado o teste de unidades, o desenvolvedor poderá melhor focar-se no que deve ser implementado para ser aprovado no teste. Nada estranho é adicionado (KIS). Estando o código completo, este pode ser testado em unidade imediatamente, e, dessa forma, prover, instantaneamente, feedback para os desenvolvedores.



Extreme Programming - XP

Codificação: Um conceito-chave na atividade de codificação (e um dos mais discutidos aspectos da XP) é a programação em dupla. Isso fornece um mecanismo para resolução de problemas em tempo real (**duas cabeças normalmente funcionam melhor do que uma**) e garantia da qualidade em tempo real (**o código é revisto à medida que é criado**).



Extreme Programming - XP

Testes: Já foi observado que a criação de testes de unidade, antes de começar a codificação, é um elemento-chave da abordagem XP. Os testes de unidade criados devem ser implementados usando-se uma metodologia que os capacite a ser automatizados (assim, poderão ser executados fácil e repetidamente)



Extreme Programming - XP

Testes: Os testes de aceitação da XP, também denominados testes de cliente, são especificados pelo cliente e mantêm o foco nas características e na funcionalidade do sistema total que são visíveis e que podem ser revistas pelo cliente



Scrum

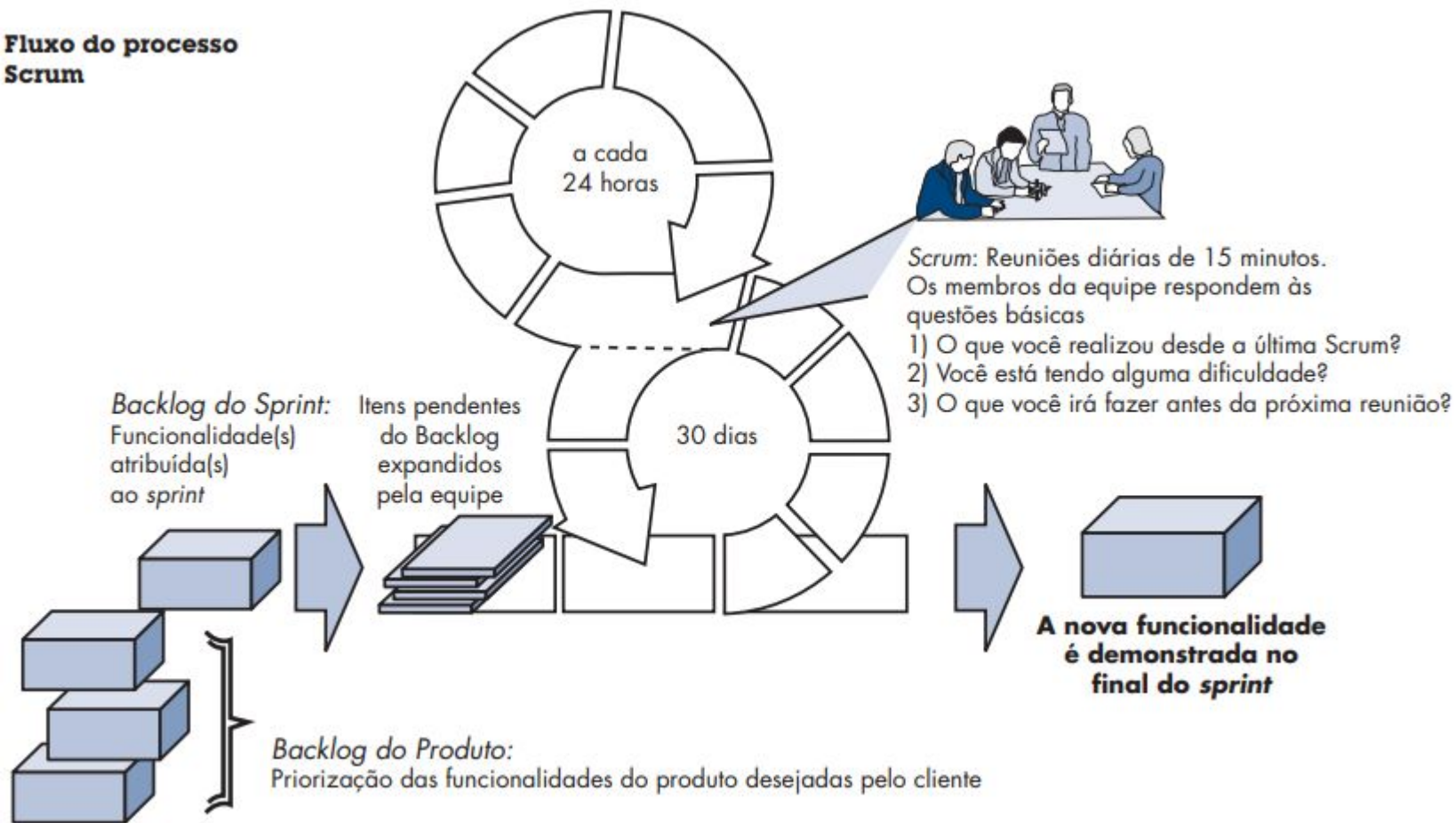


Scrum

Scrum (o nome provém de uma atividade que ocorre durante a partida de rugby) é um método de desenvolvimento ágil de software concebido por Jeff Sutherland e sua equipe de desenvolvimento no início dos anos 1990. Mais recentemente, foram realizados desenvolvimentos adicionais nos métodos gráficos Scrum por Jeff Sutherland e Ken Schwaber .

Scrum - rugby: Um grupo de jogadores faz uma formação em torno da bola e seus companheiros de equipe trabalham juntos (às vezes, de forma violenta!) para avançar com a bola em direção ao fundo do campo (Ex: https://www.youtube.com/watch?v=8UWldOwniJ4&ab_channel=WorldRugby)

Fluxo do processo Scrum





Scrum

Os princípios do Scrum são consistentes com o manifesto ágil e são usados para orientar as atividades de desenvolvimento dentro de um processo que incorpora as seguintes atividades estruturais: **requisitos, análise, projeto, evolução e entrega.**



Scrum

Em cada atividade metodológica, ocorrem tarefas a realizar dentro de um padrão de processo (discutido no parágrafo a seguir) chamado ***sprint***. O trabalho realizado dentro de um ***sprint*** é adaptado ao problema em questão e definido, e muitas vezes modificado em tempo real, pela equipe Scrum



Scrum

O Scrum enfatiza o uso de um conjunto de padrões de processos de software que provaram ser eficazes para projetos com prazos de entrega apertados, requisitos mutáveis e críticos de negócio.

- Registro pendente de trabalhos (*Backlog*)
- Urgências (corridas de curta distância) *sprints*
- Reuniões Scrum
- Demos



Scrum

Registro pendente de trabalhos (*Backlog*) — uma lista com prioridades dos requisitos ou funcionalidades do projeto que fornecem valor comercial ao cliente. Os itens podem ser adicionados a esse registro em qualquer momento (é assim que as alterações são introduzidas). O gerente de produto avalia o registro e atualiza as prioridades conforme requisitado.



Scrum

Urgências (corridas de curta distância) sprints — consistem de unidades de trabalho solicitadas para atingir um requisito estabelecido no registro de trabalho (*backlog*) e que precisa ser ajustado dentro de um prazo já fechado (janela de tempo)(1 a 4 semanas).



Scrum

Reuniões Scrum: são reuniões curtas (tipicamente 15 minutos), realizadas diariamente pela equipe Scrum. São feitas três perguntas-chave e respondidas por todos os membros da equipe :

- O que você realizou desde a última reunião de equipe?
- Quais obstáculos está encontrando?
- O que planeja realizar até a próxima reunião da equipe?



Scrum

Reuniões Scrum:

Um líder de equipe, chamado ***Scrum master***, conduz a reunião e avalia as respostas de cada integrante. A reunião Scrum, realizada diariamente, ajuda a equipe a revelar problemas potenciais o mais cedo possível. Ela também leva à “socialização do conhecimento” e, portanto, promove uma estrutura de equipe auto-organizada.



Scrum

Demos — entrega do incremento de software ao cliente para que a funcionalidade implementada possa ser demonstrada e avaliada pelo cliente. É importante notar que a demo pode não ter toda a funcionalidade planejada, mas sim funções que possam ser entregues no prazo estipulado.

Dynamic Systems Development Method (DSDM)



Método de Desenvolvimento de Sistemas Dinâmicos

É uma abordagem de desenvolvimento de software ágil que “oferece uma metodologia para construir e manter sistemas que atendem restrições de prazo apertado através do uso da **prototipagem incremental** em um ambiente de projeto controlado.



Método de Desenvolvimento de Sistemas Dinâmicos

A filosofia DSDM baseia-se em uma versão modificada do princípio de Pareto — 80% de uma aplicação pode ser entregue em 20% do tempo que levaria para entregar a aplicação completa (100%)



Método de Desenvolvimento de Sistemas Dinâmicos

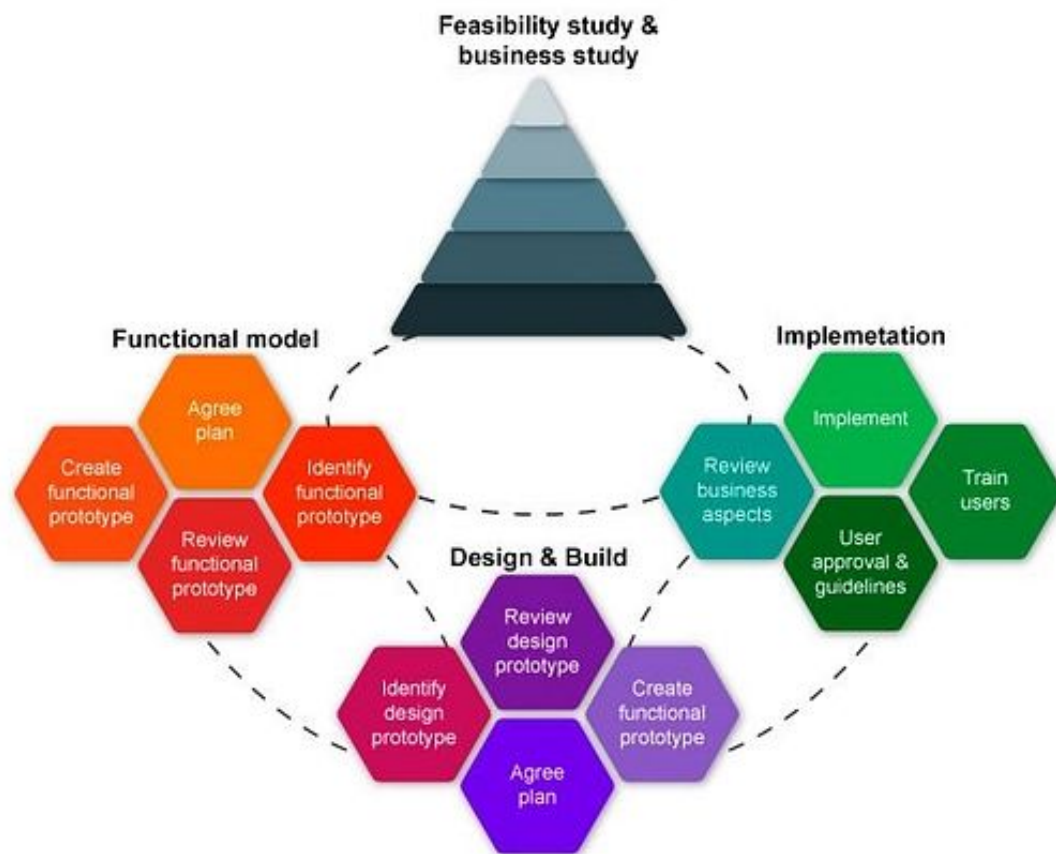
O DSDM é um processo de software iterativo em que cada iteração segue a regra dos 80%. Ou seja, somente o trabalho suficiente é requisitado para cada incremento, para facilitar o movimento para o próximo incremento.

www.dsdm.org



Método de Desenvolvimento de Sistemas Dinâmicos

Ciclo de vida DSDM que define três ciclos iterativos diferentes, precedidos por duas atividades de ciclo de vida adicionais:





Método de Desenvolvimento de Sistemas Dinâmicos

Estudo da viabilidade — estabelece os requisitos básicos de negócio e restrições associados à aplicação a ser construída e depois avalia se a aplicação é ou não um candidato viável para o processo DSDM.



Método de Desenvolvimento de Sistemas Dinâmicos

Estudo do negócio — estabelece os **requisitos funcionais** e de informação que permitirão à aplicação agregar valor de negócio; define também a arquitetura básica da aplicação e identifica os requisitos de facilidade de manutenção para a aplicação.



Método de Desenvolvimento de Sistemas Dinâmicos

Iteração de modelos funcionais — produz um conjunto de protótipos incrementais que demonstram funcionalidade para o cliente. Durante esse ciclo iterativo, o objetivo é juntar requisitos adicionais ao se obter feedback dos usuários, conforme testam o protótipo.



Método de Desenvolvimento de Sistemas Dinâmicos

Iteração de projeto e desenvolvimento — revisita protótipos desenvolvidos durante a iteração de modelos funcionais para assegurar-se de que cada um tenha passado por um processo de engenharia para capacitá-los a oferecer, aos usuários finais, valor de negócio em termos operacionais. Em alguns casos, a iteração de modelos funcionais e a iteração de projeto e desenvolvimento ocorrem ao mesmo tempo.



Método de Desenvolvimento de Sistemas Dinâmicos

Implementação — aloca a última versão do incremento de software (um protótipo “operacionalizado”) no ambiente operacional. Deve-se notar que:

- o incremento pode não estar 100% completo;
- alterações podem vir a ser solicitadas conforme o incremento seja alocado.

Em qualquer dos casos, o trabalho de desenvolvimento do DSDM continua, retornando-se à atividade de iteração do modelo funcional.

Feature Driven Development (FDD)

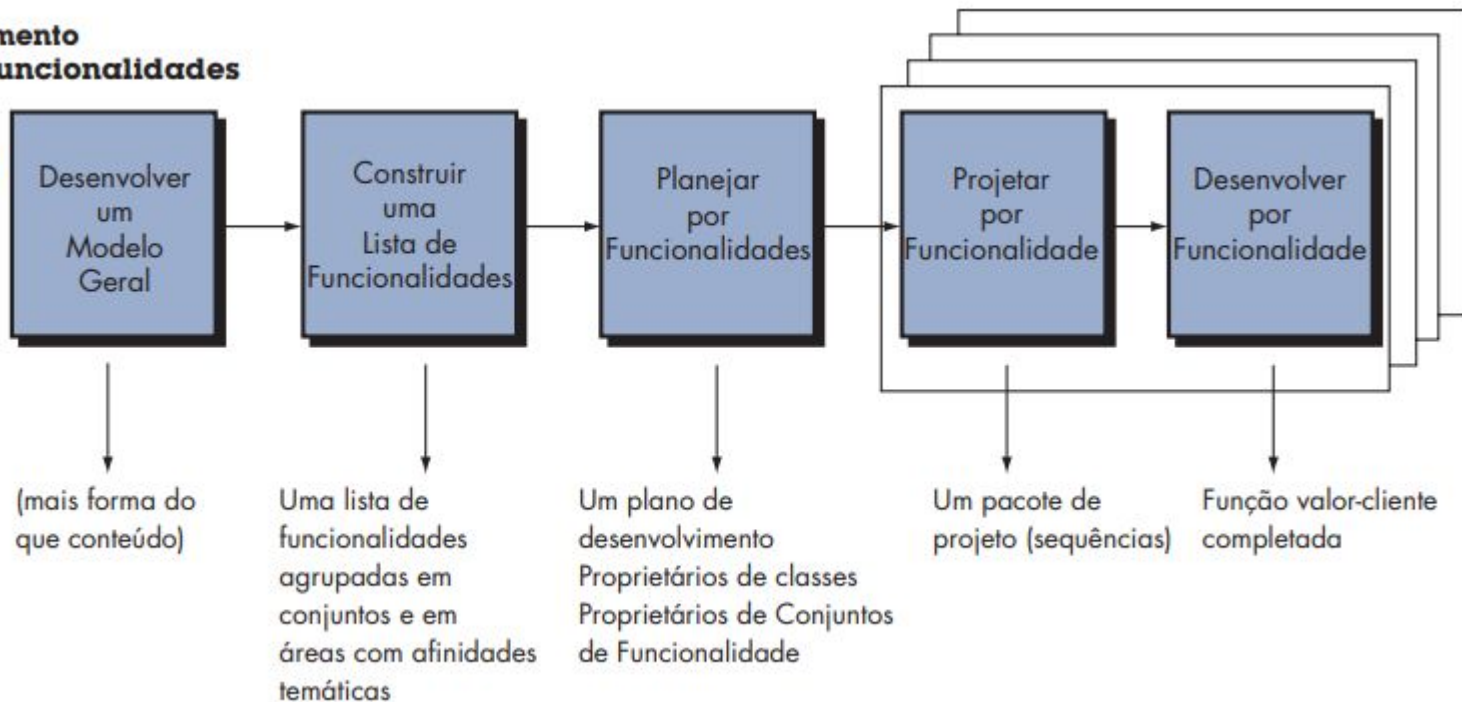


Desenvolvimento Dirigido a Funcionalidades (FDD)

Foi concebido originalmente por Peter Coad e seus colegas como um modelo de processos prático para a engenharia de software orientada a objetos criado entre 1997 e 1999, durante um projeto de desenvolvimento de software para um grande banco em Cingapura.

FDD foi formalmente apresentado e descrito ao público no livro "Java Modeling in Color with UML", de Peter Coad, Eric Lefebvre e Jeff De Luca, publicado em 1999.

**Desenvolvimento
dirigido a funcionalidades
(Coa99)
(com
permissão)**





Desenvolvimento Dirigido a Funcionalidades (FDD)

Como outras abordagens ágeis, o FDD adota uma filosofia que:

- Enfatiza a colaboração entre pessoas da equipe FDD;
- Gerência problemas e complexidade de projetos utilizando a decomposição baseada em funcionalidades, seguida pela integração dos incrementos de software;
- Comunicação de detalhes técnicos usando meios verbais, gráficos e de texto.



Desenvolvimento Dirigido a Funcionalidades (FDD)

No contexto do FDD, funcionalidade é uma função valorizada pelo cliente passível de ser implementada em duas semanas ou menos.



Desenvolvimento Dirigido a Funcionalidades (FDD)

A ênfase na definição de funcionalidades gera os seguintes benefícios:

- Como as funcionalidades formam pequenos blocos que podem ser entregues, os usuários podem descrevê-las mais facilmente; compreender como se relacionam entre si mais prontamente; e revisá-las melhor em termos de ambiguidade, erros ou omissões.
- As funcionalidades podem ser organizadas em um agrupamento hierárquico relacionado com o negócio.



Desenvolvimento Dirigido a Funcionalidades (FDD)

A ênfase na definição de funcionalidades gera os seguintes benefícios:

- Como uma funcionalidade é o incremento de software do FDD que pode ser entregue, a equipe desenvolve funcionalidades operacionais a cada duas semanas.
- Pelo fato de o bloco de funcionalidades ser pequeno, seus projeto e representações de código são mais fáceis de inspecionar efetivamente.



Desenvolvimento Dirigido a Funcionalidades (FDD)

A ênfase na definição de funcionalidades gera os seguintes benefícios:

- O planejamento, cronograma e acompanhamento do projeto são guiados pela hierarquia de funcionalidades, em vez de um conjunto de tarefas de engenharia de software arbitrariamente adotado.



Desenvolvimento Dirigido a Funcionalidades (FDD)

Coad e seus colegas sugerem o seguinte modelo para definir uma funcionalidade:

<ação> o <resultado> <por|para quem| de | para que> um <objeto>



Desenvolvimento Dirigido a Funcionalidades (FDD)

Um <objeto> é “uma pessoa, local ou coisa (inclusive papéis, momentos no tempo ou intervalos de tempo ou descrições parecidas com aquelas encontradas em catálogos)”.



Desenvolvimento Dirigido a Funcionalidades (FDD)

Exemplos de funcionalidades para uma aplicação de comércio eletrônico poderiam ser:

Adicione o produto ao carrinho

Mostre as especificações técnicas do produto

Armazene as informações de remessa para o cliente



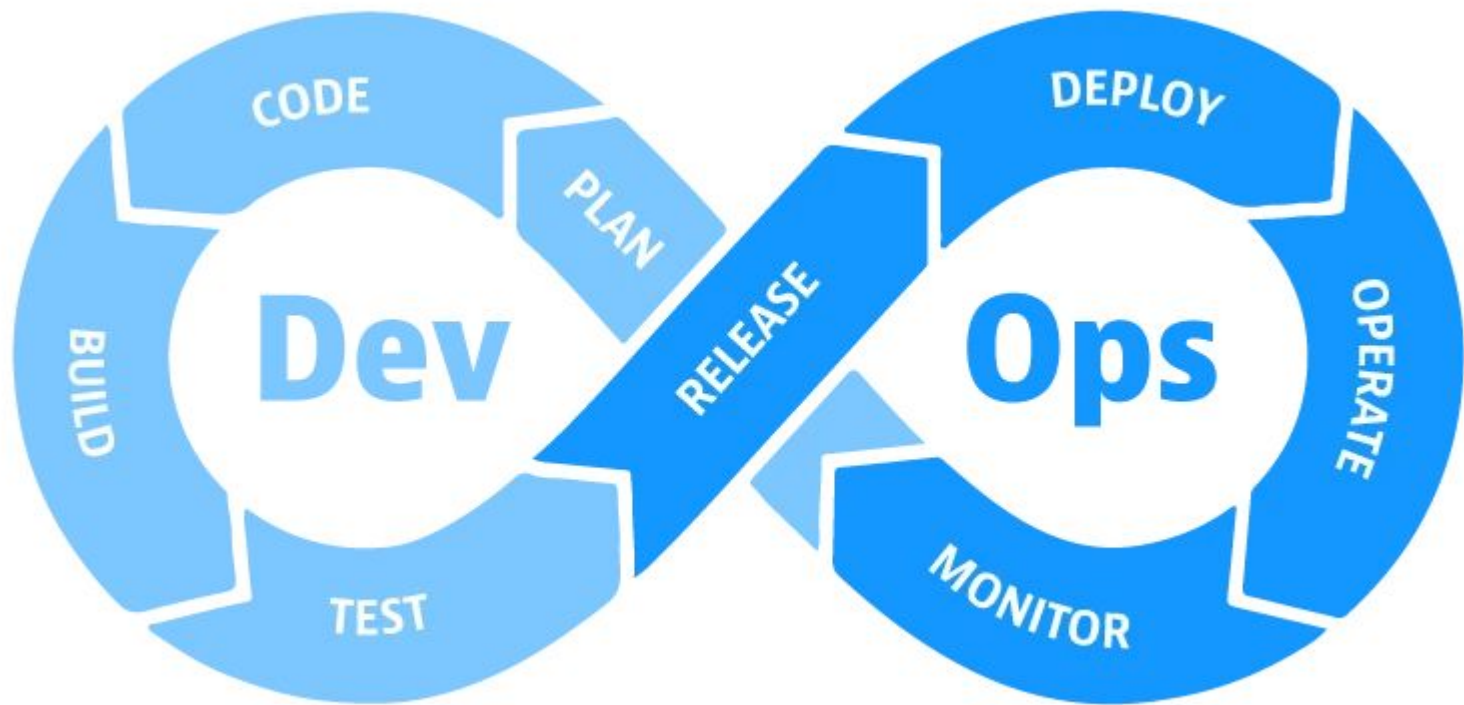
Desenvolvimento Dirigido a Funcionalidades (FDD)

Um conjunto de funcionalidades agrupa funcionalidades em categorias correlacionadas por negócio e é definido com:

<ação> um <objeto>

Por exemplo: **Fazer a venda** de um **produto** é um conjunto de funcionalidades que abrangeria os fatores percebidos anteriormente e outros

DevOps





DevOps

O DevOps foi criado por Patrick DeBois para combinar Desenvolvimento (*Development*) e Operações (*Operations*) em meados de 2007 - 2009 . O - DevOps tenta aplicar os princípios do desenvolvimento ágil e do enxuto a toda a cadeia logística de software.



DevOps

Planejar (*Plan*): Nesta é definido os objetivos do negócio, os requisitos do projeto e as funcionalidades a serem desenvolvidas. O planejamento é colaborativo, envolvendo desenvolvedores, analistas de qualidade, operações e stakeholders para garantir que todos estejam alinhados.

Compilar (*Build*): O código-fonte escrito é compilado e empacotado em um artefato executável (um "*build*"). Esta fase é automatizada através de ferramentas de Integração Contínua (CI), que compilam o código automaticamente sempre que uma nova alteração é enviada ao repositório.



DevOps

Codificar (*Code*): Os desenvolvedores escrevem o código-fonte da aplicação com base no que foi planejado. A colaboração é fundamental aqui, geralmente utilizando sistemas de controle de versão como o Git, que permitem que vários desenvolvedores trabalhem simultaneamente no mesmo projeto de forma organizada.

Testar (*Test*): O build gerado passa por uma série de testes automatizados (testes unitários, de integração, de desempenho, etc.) para garantir a qualidade e identificar falhas o mais cedo possível. A automação dos testes é crucial para manter a agilidade do ciclo.



DevOps

Lançar (*Release*): Uma vez que o software passa em todos os testes, ele está pronto para ser lançado. Nesta etapa, o artefato é preparado e versionado para a implantação. É a fase que formaliza que uma nova versão está pronta para ir para o ambiente de produção.

Implantar (*Deploy*): O pacote da nova versão é implantado (instalado) no ambiente de produção. Essa fase também é altamente automatizada, utilizando práticas de Entrega Contínua (CD - Continuous Delivery) ou Implantação Contínua (CD - Continuous Deployment), o que permite lançamentos rápidos e com baixo risco



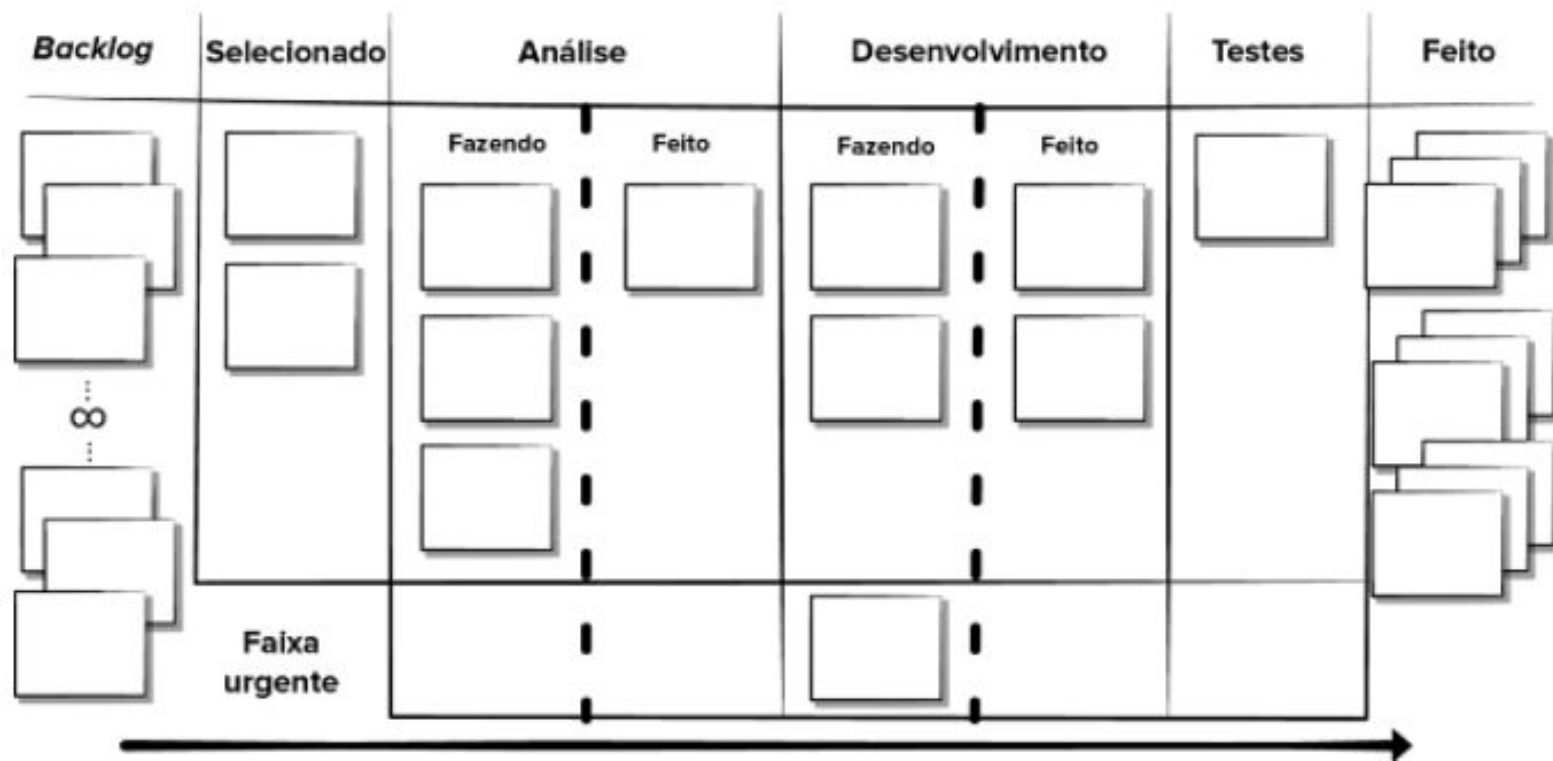
DevOps

Operar (*Operate*): Com a aplicação em produção, a equipe de operações gerencia a infraestrutura e garante que o software funcione de maneira estável e eficiente para o usuário final. Práticas como Infraestrutura como Código (IaC) são comuns aqui para automatizar o gerenciamento do ambiente.

Monitorar (*Monitor*): A aplicação e a infraestrutura são monitoradas continuamente para coletar dados sobre desempenho, uso e possíveis erros. As informações coletadas (*feedback*) são essenciais para identificar problemas e fornecer insights que alimentarão a fase de Planejamento do próximo ciclo, fechando o *loop*.



Kanban





Kanban

O Kanban foi criado pela Toyota como um sistema de controle de produção para manufatura, sem qualquer relação com software, na década de 1940.

A publicação do livro de David J. Anderson, "Kanban: Successful Evolutionary Change for Your Technology Business", em 2010, foi o marco que solidificou o Kanban como uma metodologia ágil formal e completa para o desenvolvimento de software.



Kanban

Kanban é uma técnica visual usada para gerenciar fluxos de trabalho e tarefas em processos de desenvolvimento. É amplamente utilizado em metodologias ágeis e fornece uma maneira clara de visualizar o progresso das atividades.



Kanban

Backlog:

- Todas as tarefas que ainda não foram selecionadas para execução.
- Representa o trabalho que ainda está aguardando para ser iniciado.

Fluxo de Trabalho (*Workflow*):

- O quadro Kanban é dividido em colunas que representam diferentes estágios do processo, como "Análise", "Desenvolvimento", "Testes", e "Feito".
- As tarefas são movidas da esquerda para a direita à medida que avançam no processo.



Kanban

Limites de Trabalho em Progresso (WIP):

- Limita o número de tarefas que podem estar em progresso em cada estágio. Isso ajuda a manter o foco e evitar sobrecarga de trabalho.

Faixa Urgente:

- Uma área especial no quadro para priorizar tarefas que precisam de atenção imediata.

Cartões de Tarefas:

- Cada tarefa é representada por um cartão que se move através das colunas do quadro conforme seu progresso.



Kanban

Benefícios:

- Visibilidade: Fornece uma visão clara do progresso do trabalho.
- Identificação de Gargalos: Facilita a identificação de onde o trabalho está acumulando.
- Priorização Visual: As tarefas mais urgentes podem ser facilmente destacadas.
- Limite de Trabalho em Progresso: Aumenta a eficiência ao limitar o trabalho em andamento.



Kanban

Desvantagens:

- Falta de Enfoque em Prazos: O Kanban é mais focado no fluxo do que em prazos específicos, o que pode ser um problema para projetos com cronogramas rígidos.
- Dependência da Autodisciplina: Kanban exige que a equipe siga as regras e mantenha o controle do fluxo de trabalho. A falta de autodisciplina pode prejudicar o processo.
- Visibilidade Parcial do Planejamento: Embora o quadro mostre o que está sendo feito no momento, ele não fornece uma visão ampla de planejamento futuro, como um backlog de longo prazo.



Exercício

Desenvolva os exercícios da lista.



Bibliografia

BEZERRA, Eduardo. Princípios de análise e projeto de sistemas com UML. 2. ed. rev. atual. Rio de Janeiro, RJ: Campus, 2007. 369 p.

LARMAN, Craig. Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo. 3. ed., reimpr. Porto Alegre: Bookman, 2007. 695 p. ISBN 978-85-60031-52-8

WAZLAWICK, Raul Sidnei. Análise e projeto de sistemas de informação orientados a objetos. 2. ed. rev. e atual. Rio de Janeiro: Elsevier, 2011. 330 p. (Série SBC, Sociedade Brasileira de computação) ISBN 978-85-352-3916-4

PRESSMAN, Roger S. Engenharia de software. 6. ed. Rio de Janeiro, RJ: McGraw Hill, 2006. 720 p