

Métodos Ágeis

Métodos Ágeis

- Um ponto importante para o sucesso de um produto de *software* é ser lançado antes da concorrência.
- Por si, não garante o sucesso, mas é sabido que os clientes hesitam em trocar um produto que já usam por outro recente.

Métodos Ágeis

- A partir dos anos 90 surge insatisfação com o desenvolvimento orientado a planos, levando à criação dos métodos ágeis.
- Esta nova abordagem, com o seu desenvolvimento mostrou ser a que melhor permite entregar *software* com tamanha restrição de tempo e atendendo a outros parâmetros como custo e confiabilidade, além de lidar de maneira mais fácil com as mudanças.

Manifesto Ágil

Manifesto para o 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.”

Kent Beck

Mike Beedle

Arie van Bennekum

Alistair Cockburn

Ward Cunningham

Martin Fowler

James Grenning

Jim Highsmith

Andrew Hunt

Ron Jeffries

Jon Kern

Brian Marick

Robert C. Martin

Steve Mellor

Ken Schwaber

Jeff Sutherland

Dave Thomas

Métodos Ágeis

Os métodos ágeis possuem alguns princípios:

- entrega incremental e contínua (modelo iterativo incremental);
- envolvimento do cliente;
- foco nas pessoas, e não no processo;
- adaptabilidade a mudanças;
- atenção à qualidade e uso de técnicas de excelência;
- simplicidade → redução de complexidade do sistema.

Métodos Ágeis

Riscos:

- cliente não participar do processo;
- não envolvimento dos membros do time;
- dificuldade em priorizar mudanças devido a conflitos entre *stakeholders*;
- dificuldade em manter a simplicidade;
- resistência das organizações em adaptar seus processos.

Ágil × Rápido

- Ágil significa dar respostas rapidamente às mudanças.
- Os metodos ágeis não garantem que o *software* será entregue em menos tempo.
- Mas visam garantir que o processo se adaptará melhor às mudanças.

Métodos Ágeis × Métodos Orientados a Planos

- Os métodos orientados a planos, geralmente, exigem grande burocracia.
- Por exemplo, um projeto pode estar avaliado em 40% de conclusão, porém nada foi implementado.
- Nos métodos ágeis, o esforço de documentação é substituído, em parte, pela criação do *software*.

Documentação

- Documentação extensa não garante que não haverá mudanças.
- Pelo contrário, as mudanças levarão à reescrita da documentação.
- Se não devidamente alterada, criará inconsistência entre documentação e *software* implementado.

Documentação

- Os métodos ágeis, no entanto, não são caóticos. Eles estabelecem, e necessitam de regras bem definidas e disciplina ao cumpri-las.
- A documentação, nos métodos ágeis, só é criada quando agrega valor.
- Um exemplo de documentação com altíssimo valor agregado é os códigos de teste automatizado.

Planejamento

- A gerência de projetos tradicional estabelece planejamento detalhado tanto a curto quanto a médio e longo prazo.
- Já na gerência de projetos ágeis, o detalhamento é feito apenas no planejamento de curto prazo. No médio prazo, é feito um planejamento de alto nível e, no longo prazo, é esboçada somente a visão do que espera-se ser atingido.
- Deste modo, as respostas às mudanças são mais rápidas e geram menos impacto (retrabalho).

Métodos Ágeis

- Há diversos métodos ágeis.
- Dentre estes, os mais famosos são o eXtreme Programming (XP) e o Scrum.

eXtreme Programming

O XP compreende as seguintes práticas:

- programação em pares;
- desenvolvimento orientado a testes (TDD);
- planejamento incremental;
- cliente no local;
- refatoração;
- integração contínua;
- pequenos *releases*;
- projeto simples;
- padrões de codificação;
- propriedade coletiva do código;
- metáfora do sistema;
- semana de trabalho de 40 horas.

Programação em pares

- Na programação em pares, dois programadores trabalham na mesma estação de trabalho.
- Um programador fica como "piloto", trabalhando diretamente no teclado.
- O outro programador fica como "copiloto", auxiliando o primeiro.

Programação em pares

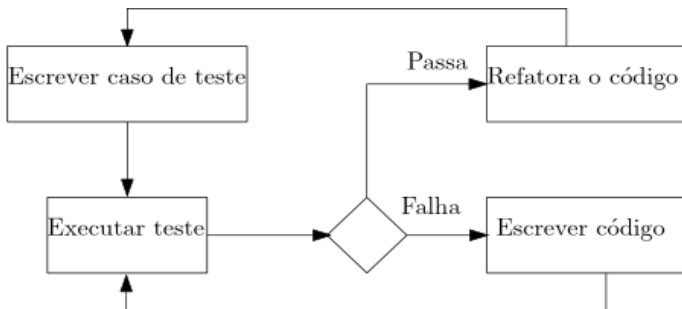
Henrik Kniberg, no livro *Scrum and XP from the Trenches* (2ed.) fala das seguintes conclusões que teve ao utilizar programação em pares:

- melhora na qualidade do código;
- melhora o foco da equipe no escopo;
- a resistência à programação em pares tem motivação no fato de muitos programadores nunca terem experimentado-a;
- é exaustiva e não deve ser utilizada todos os dias;
- alguns programadores não ficam confortáveis com esta prática;
- revisão de código é uma alternativa válida à programação em pares;
- o copiloto (ou navegador) deve ter um computador à disposição para consultar documentação, etc.;
- não deve ser forçada à equipe.


Desenvolvimento orientado a testes (TDD)

- No desenvolvimento dirigido a testes, testes e desenvolvimento são realizados de maneira intercalada.
- O código é desenvolvido de maneira incremental e sempre para corrigir uma falha em um teste.

Test Driven Development



1

¹Na refatoração o código deve obviamente passar nos testes. 

Desenvolvimento orientado a testes (TDD)

Henrik Kniberg, faz algumas ponderações sobre o TDD:

- TDD é difícil de aprender, porém "viciante";
- efeitos profundos no desenho do sistema;
- demora para colocar em novo produto, porém com rápido retorno sobre investimento;
- garantir o tempo necessário para escrever testes fáceis (ferramentas corretas, educação, prover as classes utilitárias e base certas).
- garantir que cada *feature* chave tem ao menos um teste de aceitação fim-a-fim;
- garantir que qualquer código complexo ou crítico é coberto por testes de unidade;
- saber exatamente qual parte do código não está coberta por testes;
- não deixar para escrever testes depois.

Planejamento incremental

- No XP, assim como nos demais métodos ágeis, o planejamento deve ser realizado incrementalmente.
- Decisões de desenho e organização devem ser realizadas de acordo com a necessidade.

Cliente no local

- O XP espera maior envolvimento do cliente.
- Cliente aqui significa aquele que define os requisitos.
- Para o desenvolvimento de produto de *software*, onde o responsável pelos requisitos (gerente de produto) faz parte da empresa desenvolvedora, esta parte é facilitada.
- Para o desenvolvimento de projeto de *software* pode ser mais complicado um maior envolvimento do cliente.

Refatoração

- Refatoração é a reescrita de código com o intuito de melhorá-lo.
- Um código melhor geralmente significa código mais legível.
- Outros elementos presentes na refatoração de código são:
 - a eliminação de funções, classes e objetos desnecessários;
 - divisão de funções em mais funções;
 - uso de padrões de projeto e melhorias de arquitetura;
 - outros.

Integração contínua

- Toda nova entrega deve integrar-se ao *software* já entregue.
- Ao subir o novo código ao repositório, o sistema de integração contínua faz seu *build* automático, executa testes automatizados e realiza o *deploy*.
- Alguns sistemas não fazem o *deploy* automatizado. Para estes casos, os sistemas geralmente entregam o pacote já pronto para que o *deploy* seja feito.
- Acaba com o problema do "na minha máquina funciona".

Pequenos *releases*

- O método ágil pressupõe um modelo interativo incremental, então faz sentido que as *releases* sejam pequenas.
- Assim, haverá mais entregas e consequentemente mais testes e mais *feedback* dos usuários.

Projeto simples

- Não apenas o código deve ser simples e sempre refatorado, mas o projeto como um todo.
- Deve-se sempre escolher soluções que diminuam a complexidade.

Padrões de codificação

- Em projetos onde há mais de um desenvolvedor, é interessante que a organização defina um padrão de codificação.
- Isto significa que os códigos desenvolvidos pelos mais diferentes programadores serão de mais fácil entendimento pelos demais.
- Evita-se assim o problema de pedaços de código onde só um determinado programador sabe mexer.

Propriedade coletiva do código

- O XP estabelece que não há propriedade de parte do código por um determinado programador.
- Toda a equipe é "dona" do código e pode modificá-lo.
- O Spotify utiliza modelo de "*internal open source*" onde os desenvolvedores podem propor códigos aos mais diferentes projetos da empresa através de um servidor interno de

Metáfora do sistema

- A comunicação entre clientes e desenvolvedores devem ser o mais simples o possível.
- No livro Extreme Programming Explained, Kent Beck define metáfora do sistema como "uma história que qualquer um - clientes, programadores e gerentes - podem contar sobre o funcionamento do programa".

Semana de trabalho de 40 horas

- Quando se fala de semana de trabalho de 40 horas o importante não é apenas a quantidade de horas, mas a qualidade destas horas.
- Muitas reuniões, retrabalho e falhas de comunicação podem impedir que os programadores trabalhem da melhor maneira.
- Como a produção de *software* de alta qualidade é um trabalho mental, simplesmente colocar mais horas também não resolverá problema algum.

Scrum

- O Scrum é método ágil que define um *framework* para a organização de um projeto ágil.
- Diferente do XP, por exemplo, o Scrum não se baseia em um conjunto de técnicas, mas em definir papéis, artefatos, eventos e regras.
- Segundo Henrik Kniberg, no Scrum original, Jeff Sutherland utilizava as técnicas do XP, mas foi convencido por Ken Schwarber a manter o modelo mais simples, colocando a responsabilidade das equipes em escolher o que deveriam usar.

Scrum

O Scrum define os seguintes papéis:

- *Development Team*;
- *Scrum Master*;
- *Product Owner*.

Development Team

- auto-organizável → toma as decisões de como transformar o *Product Backlog* (conjunto de requisitos) em um produto funcional;
- possui todas as características, como um time, necessárias para criar um incremento do produto;
- todos são chamados de desenvolvedores independente do trabalho que realizam;
- não há sub-times;
- possui entre 3 e 7 integrantes (sem incluir o *Scrum Master* e o *Product Owner*).

Scrum Master

- O *Scrum Master* é o guardião das regras do *Scrum* e responsável por garantir o respeito às mesmas.
- Faz a ligação entre o time, o *Product Owner* e o público externo.

Product Owner

- Responsável por decidir quais funcionalidades serão implementadas e em qual ordem.
- Responsável por gerenciar o *Product Backlog* (conjunto de requisitos).

Artefatos do *Scrum*

Os artefatos utilizados pelo *Scrum* são:

- *Product Backlog*;
- *Sprint Backlog*.

Product Backlog

- O *Product Backlog* é lista priorizada contendo os requisitos e características do produto a ser implementado.
- Ele nunca está completo e é alterado a medida que se vai entendendo o que será construído.
- Pode ser utilizado por múltiplos times que estejam construindo o mesmo produto.
- O *Product Owner* é o dono do *Product Backlog*.

Product Backlog

Um *Product Backlog* pode conter:

- características;
- funcionalidades;
- recursos;
- *bugs*;
- entre outros.

Product Backlog

Os itens do *Product Backlog* são divididos em:

- a fazer;
- em execução;
- prontos.

Product Backlog

Cada *Product Backlog Item* (PBI) pode ter os seguintes campos:

- identificador;
- nome;
- importância;
- estimativa;
- estado.

Outros campos podem ser incluídos se necessários.

Product Backlog

ID	nome	imp.	estim.	estado
36	Enquanto vendedor, desejo listar as vendas do último mês para ter um controle contábil.	21	2	a fazer
5	Enquanto cliente, desejo inserir itens no carrinho para comprá-los.	34	65	em execução
16	Adaptar a <i>interface</i> de usuário para cegos.	8	34	a fazer
3	Criptografar os dados bancários.	34	21	feito

Sprint Backlog

- O *Sprint Backlog* é criado a partir do *Product Backlog*.
- O dono do *Sprint Backlog* é o *Development Team*.
- Ele define as atividades que serão realizadas para que um determinado PBI seja implementado/realizado.

Sprint Backlog

ID	nome	tarefa	resp.	estim	estado
36	Enquanto vendedor, desejo listar as vendas do último mês para ter um controle contábil.	criar <i>interface</i>	Phelipe	16h	feito
		implem. banco	Cleisson	12h	feito
		implem. código	Diêgo	6h	feito
		criar <i>thumbnails</i>	Jesse	4h	feito
		corrig. msg. erro	Mário	2h	exec.

Scrum

- O Scrum é o modelo iterativo incremental implementado de maneira radical.
- Isto significa que o processo de desenvolvimento utilizando Scrum é dividido em iterações, chamadas *Sprints*.
- Cada *Sprint* tem um tempo definido (entre 2 e 4 semanas).

Sprint

- Toda *Sprint* tem um objetivo e terá sempre como resultado *software* funcional, um protótipo ou um desenho arquitetural.
- No caso do primeiro, este *software* deve já estar integrado ao que foi feito anteriormente e deve ser colocado em ambiente de produção.
- Protótipos e desenhos arquiteturais servem para embasar *sprints* futuras.

Sprint

As *Sprints* podem ser de três tipos:

- funcional → quando seu objetivo é implementar funcionalidades para o usuário final;
- performance e confiabilidade → quando seu objetivo é melhorar performance, segurança, confiabilidade e eficiência do sistema;
- suporte → quando seu objetivo é atividades auxiliares, como desenvolver *software* de infraestrutura ou projetar a arquitetura do sistema.

Sprint

Durante uma *Sprint*:

- não há mudanças que ameacem os objetivos da *Sprint*;
- metas de qualidade não diminuem;
- o escopo pode ser clarificado e renegociado entre o *Product Owner* e o time (se necessário).

Sprint Planning

- O pontapé inicial da *Sprint* é chamado *Sprint Planning*.
- É realizado em, no máximo, 8 horas (para uma *Sprint* de 4 semanas) e envolve a equipe toda.
- Deve tratar 2 tópicos:
 - o que pode ser feito nesta *Sprint*?
 - como o trabalho escolhido será realizado?

Daily Scrum

- Durante cada dia de trabalho é realizado, no seu início, o *Daily Scrum*.
- O *Daily Scrum* é uma reunião de 15 minutos entre o time de desenvolvimento e o *Scrum Master* com o intuito de sincronizar o trabalho e planejá-lo para as próximas 24 horas.

Daily Scrum

- A reunião é realizada em pé e cada integrante diz o que fez no dia anterior e o que fará neste.
- Quando um membro do time possui algum problema, ele o trás ao restante dos integrantes, mas na reunião não é discutido como será resolvido.
- A solução será discutida após a reunião e apenas entre as pessoas que realmente possam resolvê-lo.

Sprint Review

- A *Sprint Review* é realizada ao fim da *Sprint* para inspecionar o incremento e adaptar o *Product Backlog* se necessário.
- A reunião pode durar até 4 horas (para uma *Sprint* de 4 semanas) e envolve o time inteiro e *stakeholders* convidados pelo *Product Owner*.
- O *Product Owner* decide se o que foi feito está correto ou não. Caso ele diga que um PBI foi feito completamente correto, então ele é considerada “feito” (“*done*”).
- Caso contrário, este item deverá ser corrigida no próximo *sprint*.
- A *Review* também serve para levantar os problemas ocorridos durante a *Sprint* e verificar soluções adotadas ou a serem adotadas.

Retrospectiva da *Sprint*

- A retrospectiva da *Sprint* ocorre após a *Review* da *Sprint* e antes do planejamento da próxima *Sprint*.
- Nela são discutidos todos acontecimentos da última *Sprint* e são elaboradas soluções para melhoria do *Scrum*.
- Tem a duração de 3 horas (para *Sprint* de 4 semanas) e participação do *Development Team* e do *Scrum Master*.

Product Backlog Grooming

- Antes de começar uma nova *Sprint*, é importante revisar o *Product Backlog*.
- Este evento é chamado de *Product Backlog Grooming*.
- São realizadas quatro atividades:
 - refinamento → os PBIs são analisados e refinados, podendo haver a criação de novos itens;
 - estimação → o *Development Team* analise cada PBI e verifica o esforço necessário para sua implementação;
 - criação → novos itens são inseridos, removidos ou modificados no *Product Backlog*;
 - priorização → os PBIs são reordenados de acordo com sua priorização.

Cancelamento de uma *Sprint*

- Apenas o *Product Owner* pode cancelar uma *Sprint*.
- Uma *Sprint* geralmente é cancelada quando seu objetivo se torna obsoleto.
- Quando uma *Sprint* é cancelada:
 - itens do *Product Backlog* dados como “feitos” são cancelados;
 - o trabalho feito, se aproveitável, geralmente é aceito;
 - os itens não terminados são reavaliados e retornam do *Product Backlog* se não obsoletos.
- Cancelamentos de *Sprints* são, em geral, bastante traumáticos para equipe e devem ser tratados com seriedade.

Riscos do Scrum

- O Scrum parte do pressuposto que a equipe é composta somente por trabalhadores dedicados que compartilham um mesmo espaço. Isto nem sempre ocorre, especialmente com o crescimento vertiginoso do trabalho remoto.
- Nem sempre todos os membros do time poderão participar do *Daily Scrum* de manhã devido a possibilidade de horários flexíveis.
- Nesses casos, a equipe deverá criar maneiras de se comunicar com maior fluidez, pois esse é o grande ponto-chave dos métodos ágeis.

Referências

- Sommerville, Ian. Software Engineering - Global Edition. 10ed. 2016. Pearson Education.
- Sommerville, Ian. Engineering Software Products: An Introduction to Modern Software Engineering. 1ed. 2021. Pearson Education.
- Kniberg, Henrik. Scrum and XP from the trenches. 2ed. 2015. C4Media.
- Beck, Kent e Andres, Cynthia. Extreme Programming Explained: Embrace Change. 2ed. 2004. Addison Wesley Professional.
- <https://www.altexsoft.com/blog/business/extreme-programming-values-principles-and-practices/>
- <https://www.altexsoft.com/blog/business/continuous-delivery-and-integration-rapid-updates-by-automating-quality-assurance/>