**FATEC JUNDIAÍ**

**ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

Isac Petinate

Larissa Narciso

Mariana Garcia

Vitor Duran

Viviane Ogata

**METODOLOGIAS ÁGEIS**

Jundiaí

2018

**SUMÁRIO**

[1 – Metodologias Ágeis - Descrição 3](#_Toc531159933)

[2 - Entendendo o Problema 4](#_Toc531159934)

[3 – METODOLOGIAS ÁGEIS 6](#_Toc531159935)

[4 – Frameworks agile 19](#_Toc531159936)

[4.1 - XP - Extreme Programming 19](#_Toc531159937)

[4.2 - FDD - Feature Driven Development 28](#_Toc531159938)

[4.3 - Kanban 34](#_Toc531159939)

[4.4 - SCRUM 45](#_Toc531159940)

[5 – Referências Bibliográficas 57](#_Toc531159941)

## 

## 1 – Metodologias Ágeis - Descrição

Os métodos ágeis são uma alternativa à gestão tradicional de projetos, eles nasceram nos braços do desenvolvimento de software, mas hoje podem ser aplicados a qualquer tipo de projeto (inclusive os que não se remetem ao software). Os métodos ágeis vêm ajudando muitas equipes a encarar a imprevisibilidades dentro de um projeto através de entregas incrementais e ciclos iterativos. As metodologias ágeis passaram a ser uma alternativa aos métodos tradicionais, também conhecidos como métodos pesados ou clássicos.

Os métodos ágeis buscam promover um processo de gerenciamento de projetos que incentiva a inspeção e adaptação frequente. É uma filosofia que acaba por incentivar o maior trabalho em equipe, a auto-organização, a comunicação frequente, o foco no cliente e a entrega de valor. Basicamente, os métodos ágeis são um conjunto de práticas eficazes que se destinam a permitir a entrega rápida e de alta qualidade do produto, tendo uma abordagem de negócios que alinha o desenvolvimento do projeto com as necessidades do cliente e os objetivos da empresa.

## 2 - Entendendo o Problema

A era da computação iniciou-se nos anos 40, e os maiores investimentos eram voltados a hardware. No início dos anos 50, passou-se a ter o domínio da tecnologia de hardware e então os investimentos se voltaram ao desenvolvimento dos sistemas operacionais o que possibilitou o surgimento das primeiras linguagens de programação de alto nível, permitindo assim que usuários pudessem se concentrar mais no desenvolvimento sem se preocuparem com questões técnicas do funcionamento do hardware. O surgimento de sistemas operacionais com multiprogramação, no início dos anos 60, possibilitou um aumento na eficiência destes, contribuindo para a queda de preço dos hardwares.

Com o desenvolvimento dos hardwares e dos sistemas operacionais, passou-se a haver a necessidade de sistemas mais complexos e maiores em substituição aos pequenos aplicativos que existiam até o momento e foi aí que se iniciou a “crise do software” pela incompatibilidade dos métodos utilizados até então com os métodos necessários para a criação desses sistemas. Em 1968 foi realizada uma conferência pelo Comitê de Ciência da NATO (North Atlantic Treaty Organization) com o nome “Engenharia de Software” e foi aonde, pela primeira vez, foi utilizado esse termo. Nessa conferência foi discutido sobre a existência de uma real crise de software, e o que se constatou foram, ao menos, os seguintes problemas:

* Cronogramas não observados;
* Projetos com tantas dificuldades que são abandonados;
* Módulos que não operam corretamente quando combinados;
* Programas que não fazem exatamente o que era esperado;
* Programas tão difíceis de usar que são descartados;
* Programas que simplesmente param de funcionar.

E foi a partir desse cenário que passou a se desenvolverem os processos e metodologias de desenvolvimento de software. Um dos primeiros processos, que surgiu em 1970, foi o que ficou conhecido como processo em cascata, ele possui sete fases e uma só poderia ser iniciada após o término da anterior, daí o nome, isso acaba causando alguns transtornos, afinal em muitas vezes o cliente gostaria de 10 fazer alguma mudança no sistema, porém descobre isso quando a fase de requerimentos já passou o que acaba deixando o projeto defasado ou mais caro por não prever de início esse tipo de mudança.

Nos anos 80, surgiu outro processo para desenvolvimento de software, o processo em espiral, que apareceu como a solução para os problemas existentes no modelo em cascata, a exemplo do “ciclo de Demming” (ciclo PDCA), esse modelo surgia com apenas quatro fases (Planejamento, Avaliação, Análise de Risco e Engenharia), o processo inicia no planejamento, vai para a avaliação, análise de risco, engenharia e, seguindo a ideia do espiral, volta para o planejamento fazendo todo o ciclo novamente, idealizando assim um modelo iterativo e incremental, permitindo que os erros ocorridos em uma fase possam ser revistos.

Ainda assim, com o passar do tempo, a complexidade dos sistemas tem aumentado ainda mais, os sistemas possuem mais usuários e tornam-se cada vez mais importantes, podendo, em muitas vezes, gerenciar a vida ou a morte de pessoas, como é o caso de softwares que calculam quantidades de medicamentos a serem aplicados em pacientes ou que monitoram o estado de saúde dos pacientes, os sistemas também controlam a economia e as armas do mundo, é possível ver isso através do pânico que foi causado pela possibilidade do “bug” do milênio no final dos anos 90.

Unindo-se aos problemas até então encontrados e à importância atual dos sistemas informatizados, surgiram alguns teóricos discordando da ideia de tratar o desenvolvimento de software como uma fábrica de produção em série, um exemplo é Cockburn que compara o desenvolvimento de software ao ato de escrever uma poesia épica em conjunto com diversas pessoas: seriam diversas pessoas, com diversos argumentos, tentando dar seu melhor sem talento, tempo ou recursos suficientes.

## 3 – METODOLOGIAS ÁGEIS

Em 2001 o termo “Metodologias Ágeis” tornou-se popular, quando um grupo de dezessete especialistas em processos de desenvolvimento de software decidiu se reunir nos EUA, para discutir maneiras de melhorar o desempenho de seus projetos.

“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 frequência, 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. Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho. O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara. Software funcional é a medida primária de progresso. Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes. Contínua atenção a excelência técnica e bom design, aumento da agilidade. Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito. As melhores arquiteturas, requisitos e designs emergem de times auto organizáveis. Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e aperfeiçoam seu comportamento de acordo”.

Embora cada envolvido tivesse suas próprias práticas e teorias sobre como fazer um projeto de software ter sucesso, utilizando métodos como Scrum, Extreme Programming (XP) e outros, cada um com as suas particularidades, todos concordavam que, em suas experiências prévias, um pequeno conjunto de princípios sempre parecia ter sido respeitado quando os projetos davam certo.

Foi então criada a Aliança Ágil e o estabelecimento do Manifesto Ágil contento os conceitos e princípios comuns compartilhados por todos esses métodos. Desde então o termo Desenvolvimento Ágil passou a descrever abordagens de desenvolvimento que seguissem estes conceitos que implicam em valorizar:

* Indivíduos e interação entre eles são mais que processos e ferramentas:

Os softwares não são construídos por uma única pessoa, eles são construídos por uma equipe, então elas precisam trabalhar juntas (incluindo programadores, testers, projetistas e também o cliente). Processos e ferramentas são importantes, mas não são tão importantes quanto trabalhar juntos.

* Software em funcionamento mais que documentação abrangente:

A documentação deve existir para ajudar pessoas a entender como o sistema foi construído, mas é muito mais fácil entender como o funcionamento vendo o sistema funcionar do que através de diagramas que descrevem o funcionamento ou abstraem o uso.

* Colaboração com o cliente mais que negociação de contratos:

Somente o cliente pode dizer o que ele espera do software e normalmente eles não sabem explicar exatamente o que eles esperam e ainda, eles mudam de ideia ao longo do tempo e conforme eles vêm o software funcionando. Ter um contrato é importante para definir as responsabilidades e direitos mas não deve substituir a comunicação. Trabalhos desenvolvidos com sucesso têm constante comunicação com o cliente para entender suas necessidades e ajuda-los a descobrir a melhor forma de expressa-las.

* Responder a mudanças mais que seguir um plano:

Mudanças são uma realidade no ambiente de negócios e elas acontecem por inúmeras razões: as regras e leis sofrem alterações, as pessoas mudam de ideia e a tecnologia evolui. O software precisa refletir essas mudanças. Um projeto de software certamente deve ter um plano mas ele deve ser flexível o suficiente para comportar as mudanças quando elas aparecerem, senão ele se torna irrelevante.

O Manifesto Ágil não rejeita os processos e ferramentas, a documentação, a negociação de contratos ou o planejamento, mas simplesmente mostra que eles têm importância secundária quando comparado com os indivíduos e interações, com o software funcionando, com a colaboração com o cliente e as respostas rápidas a mudanças e alterações.

É muito importante entender que os conceitos do manifesto ágil definem preferências e não alternativas no desenvolvimento de software, encorajando a focar a atenção em certos conceitos sem eliminar outros. Assim para seguir os conceitos ágeis deve-se valorizar mais a certas coisas do que a outras.

Simplificando: a metodologia ágil é uma ferramenta de simplificação de processos complexos, tendo como principal aplicação a utilização de frameworks (ferramentas de trabalho), como SCRUM, Kanban, XP, LEAN e FDD.

Algumas características comuns das metodologias ágeis são:

* Processo incremental (quase uma antítese do tradicional modelo de cascata);
* Colaboração do cliente;
* Adaptabilidade (cada projeto está sujeito a passar por várias modificações);
* Simplicidade;
* Feedback constante;
* Equipes pequenas (mas com alto nível técnico) etc.

Vantagens dos métodos ágeis para os clientes:

1. Agilidade

* O tempo de entrega do produto é um dos maiores benefícios dos métodos ágeis na perspectiva do cliente. O ciclo extremamente reduzido — em comparação aos outros métodos — é um atrativo que faz toda a diferença.
* Imaginemos, por exemplo, como o consumidor, potencialmente interessado em adquirir soluções desenvolvidas “na medida”, atendendo a todas as suas necessidades, receberia a notícia de que a previsão de entrega é de 10 meses.
* Ou seja, durante todo esse tempo a empresa terá que se conformar em ficar estagnada, sendo atropelada pela concorrência, adquirindo produtos de software prontos para mitigar os efeitos negativos do atraso.
* Colocando-se no lugar do cliente, qual seria a sua resposta? Valeria a pena aceitar que o processo é demorado, custa caro e é dificultoso em termos de implementação?
* Esse panorama muda completamente quando o desenvolvedor utiliza métodos ágeis, pois o prazo de entrega é consideravelmente reduzido e, de quebra, se baseia em várias entregas, em vez de uma.

1. Múltiplas entregas

* Dando continuidade ao tema, as múltiplas entregas que fazem parte do ciclo ágil permitem que o cliente adquira expectativas de como o software funcionará, muito antes de chegar à versão final.
* Outro destaque é que a equipe pode desenvolver uma versão do software a ser utilizada pelo cliente, antecipando o desenvolvimento das partes funcionais do programa, assegurando que ele veja um retorno de investimento (ROI, return on investment) quase imediato.
* Por fim, as eventuais falhas apresentadas pelo software podem ser detectadas pelo cliente e, assim, corrigidas com antecedência pelos desenvolvedores.

1. Participação no projeto

* Além da entrega contínua de versões do software, as metodologias ágeis integram o consumidor ao projeto, de modo que as suas solicitações e feedbacks sejam prontamente assimilados pela equipe.
* O diferencial dessa intensa participação do cliente está, acima de tudo, na transparência agregada ao ciclo do projeto. Isto é, o cliente fica a par do que está acontecendo, dos recursos que estão sendo desenvolvidos no momento etc.
* Em adição a isso, o cliente tem a oportunidade de estimar as novas funcionalidades em cada fase do projeto. Portanto, sempre que uma nova fase se iniciar, o cliente terá uma expectativa bastante realista dos recursos que receberá.

1. Customização do produto

* Por último, está a possibilidade de customizar o produto de acordo com as necessidades e preferências porque os métodos ágeis têm alta adaptabilidade.
* Suponha que, devido a mudanças em alguns processos do negócio, o cliente precise solicitar modificações — tanto sutis quanto radicais — de última hora, como a exclusão de recursos recentemente adicionados e a inclusão de outros até então não cogitáveis.
* Pela excelência técnica e conhecimento das práticas envolvidas nas metodologias, a equipe estará pronta para atender às solicitações e, então, entregar um produto em conformidade com as novas especificações.

Vantagens dos métodos ágeis para a equipe

1. Entregas rápidas e frequentes

* Logicamente, esse é um benefício que abrange tanto as perspectivas do cliente quanto da equipe. No caso da empresa, a maior vantagem é ter que gerenciar equipes menores e com profissionais experientes, o que facilita todo o processo.
* Na prática, as equipes são subdivididas de maneira que cada uma se responsabilize por determinada funcionalidade do produto, ou seja, cada grupo tem suas metas e responsabilidades que, ao fim de cada estágio, se integram às demais partes.
* Isso traz vantagens em dois aspectos: o foco e a qualidade final de cada entrega.
* A primeira delas é porque, simplesmente, os desenvolvedores se concentram numa quantidade limitada de atribuições — inclusive, é o que ajuda a manter o pessoal motivado.
* Em adição ao foco vem a qualidade do produto porque, com menos atribuições, os desenvolvedores têm mais tempo para aplicarem a excelência técnica no código e no design.

1. Qualidade do produto

* Em métodos tradicionais, o cliente somente é ouvido quando o produto está finalizado, correndo sérios riscos de apontar erros logo na implementação e, também, de contrariar boa parte das suas menores exigências.
* Os métodos ágeis, por outro lado, consistem nas entregas em escala semanal ou mensal, integrando o cliente ao processo de desenvolvimento — no caso, prestando auxílio por meio de feedbacks.
* Isso faz notável diferença para a qualidade final do software, visto que todas as falhas e modificações foram realizadas muito antes do último lançamento. Assim, a expectativa do cliente tende a ser atendida com incomparável eficiência.

1. Previsão de cronograma e custos

* O uso dos Sprints, que basicamente, nada mais é que uma reunião formada pelos envolvidos no projeto. Em cada Sprint é estabelecido um conjunto de atividades a serem executadas em determinado espaço de tempo (Time Box).
* Sendo assim, como em cada Sprint é definido o que será feito, é possível prever o tempo que o time levará para entregar o release, bem como planejar um cronograma para otimizar a agilidade e estimar o custo de cada recurso adicionado.
* Desse modo, a equipe define junto ao cliente quais recursos devem ser priorizados. A partir dos detalhes do Sprint, as duas partes analisam se haverá necessidade de iterações extras e quantas.

1. Mitigação de riscos

* Levando em conta a participação do cliente no processo e os constantes testes de software feitos pela equipe, os bugs e falhas que surgem durante o projeto são rapidamente identificados, seja pelo loop de feedbacks, seja pelos resultados dos testes.
* Essa vantagem muito se deve à liberdade que têm os projetos das numerosas restrições impostas pelo planejamento. Supondo que o modelo de cascata fosse seguido, os bugs seriam detectados tardiamente e, sem dúvida, levariam mais tempo para serem corrigidos.

Diferença da metodologia tradicional com a ágil

1. Metodologia tradicional

A metodologia tradicional, possuindo como técnica ou modelo mais conhecido o modelo clássico ou cascata (*waterfall*), que também é conhecido como abordagem “top-down”. O modelo foi proposto por Royce em 1970, derivado de modelos industriais de diversos segmentos de engenharias; seu principal objetivo era de estabelecer ordem e padrão em desenvolvimento de software. Foi chamado de desenvolvimento tradicional, pois é a base para diversos modelos utilizados há décadas pela indústria de software e é considerado um modelo rígido de pouca flexibilidade, adaptabilidade e versatilidade, sendo utilizado em projetos de pequeno, médio e grande porte.

Grande parte de seu alto grau de aceitação e utilização está no fato de ser um modelo orientado para documentação, seja ela de qualquer tipo, como textos, representações gráficas, simulação, diagramas, casos de usos, entre outras. A ideia principal dessa metodologia é que o software é construído baseado em uma sequência de fases, sendo que cada uma delas depende da conclusão da outra para ser iniciada, com exceção da primeira, conforme ilustra a Figura 1.



Figura 1: Ciclo de desenvolvimento utilizando metodologia tradicional

Os resultados da primeira etapa, da Análise, são concluídos, então a saída “flui” para a segunda etapa, que é a de Projeto. Logo, quando a etapa de Projeto for concluída, sua saída ou resultado “flui” para a seguinte, sendo que uma etapa nunca se inicia antes da etapa anterior ser finalizada. As atividades são agrupadas em tarefas executadas sequencialmente.

Além do modelo clássico, existem também variações que são empregadas de forma iterativa e incremental, tendo ainda uma forte linearidade no processo, caracterizado por “cascatas menores” dentro de cada iteração. No decorrer das tarefas e para a conclusão das mesmas, fazem parte da metodologia tradicional algumas premissas que a influenciam, como:

* Linearidade do modelo: as atividades são feitas em sequência e uma só se inicia quando a anterior está finalizada;
* Determinismo: tendo como base as especificações, se elas forem seguidas rigorosamente, o resultado gerado também será o correto. Tem-se a ideia “equivocada” de que o uso do determinismo é uma forma de reduzir erros e perdas de tempo;
* Especialização: como as atividades são realizadas de forma independente, podem ser executadas por especialistas, com a rígida divisão de papéis entre seus membros, como analistas, projetistas, programadores, testadores e implantadores. Cada um realiza tarefas bem diferentes e especializadas e terão seus resultados integrados para compor o software finalizado;
* Foco na execução: como as pessoas são em sua grande maioria especialistas, as tarefas são simples e determinísticas, bastando a elas somente a execução. Presume-se que, se cada membro da equipe executar corretamente a tarefa que lhe cabe, a especificação será transformada corretamente em software. Isso gera uma grande valorização dos processos e, em consequência, uma grande desvalorização das pessoas no desenvolvimento de software;
* Crescimento exponencial do custo de alteração: o custo de uma alteração tende a crescer à medida que o processo de desenvolvimento avança.

1. Metodologia Ágil

A metodologia ágil surgiu da necessidade de minimizar riscos e custos associados ao desenvolvimento de software. Para a utilização da metodologia ágil, o framework Scrum é a ferramenta mais indicada. É um framework pois facilita o uso de diversos processos e técnicas. O Scrum vem sendo utilizado como uma ferramenta para controlar e gerenciar o processo de desenvolvimento de produtos complexos que agregam maior valor para o cliente.

O framework Scrum é composto por ciclos de desenvolvimentos, mais conhecidos como *Sprints,*com duração fixa não ultrapassando mais de um mês. Ele é maleável, mas pouco adaptativo, ou seja, deve-se evitar ao máximo realizar mudanças desnecessárias que afetem o objetivo da *sprint*. Essas mudanças devem ser discutidas entre o *product owner* e a equipe de desenvolvimento.

Para a execução dos ciclos iterativos, o Scrum é composto por equipes com um número razoavelmente baixo de participantes para facilitar o gerenciamento, e, associados aos membros que formam as equipes, temos: papéis, artefatos e cerimônias.

* Papéis: em uma equipe ágil, existem três papéis: *product owner*, que é o responsável por maximizar o valor de retorno do produto para o cliente e fornecer o trabalho para o *development team*, que é formado pelos profissionais que irão trabalhar no desenvolvimento e teste de uma versão do produto. Ao final de cada iteração, o *scrum master* é o responsável por assegurar que o Scrum seja entendido e disseminado, por resolver impedimentos e ajudar o *product owner* e o *development team*;
* Artefatos: basicamente, são utilizados quatro artefatos:  
  - *Product backlog*: engloba todos os requisitos que precisam ser implementados. Ele é controlado pelo *product owner* e priorizado de acordo com o valor de negócio para o cliente;  
  - *Sprint* *backlog*: é uma lista de itens que define o objetivo da *sprint*. Esses itens são priorizados e aqueles que entram no *sprint* *backlog* são os que possuem maior valor de negócio apresentado pelo *product owner*;  
  - Definition of Done: documento que explica quais serão as definições de concluído ou pronto para as *user stories*;  
  - Burndown: gráfico que mostra o andamento do desenvolvimento do produto e do andamento da *sprint*. É utilizado para monitorar o andamento da *sprint*, indicando se as tarefas serão completadas de acordo com o planejamento.
* Cerimônias: no total são definidas cinco cerimônias:  
  - *Grooming*: cerimônia realizada para apresentação e estimativa das histórias. Deve envolver os *product owners* e o *development team*. Com base nas alterações e levando-se em conta a complexidade, nível de abstração ou alterações, são realizadas estimativas de histórias que podem ou não entrar na(s) próxima(s) *sprint(s)* dependendo das prioridades negociadas entre o *product owner* e o cliente;  
  - *Sprint* *Planning*: é a cerimônia de planejamento da *sprint* onde são definidas quais serão as *user stories* entregues. Após as definições das *user stories* que entram na *sprint*, cabe ao *development team* decidir como implementá-las durante a *sprint*, de modo que são realizadas quebras nas *user stories* em tarefas menores, chamadas de *tasks*. Nas *tasks* devem estar as atividades detalhadas que servirão como base para a conclusão da história;  
  - *Daily* Scrum; é uma reunião diária e curta (máximo de 15 minutos) do *scrum team* com o objetivo de observar o progresso do desenvolvimento do produto. Cada membro do *development team* basicamente deve informar para os demais o andamento de seu trabalho, respondendo a três questões: O que foi feito desde a última reunião? O que pretende fazer até a próxima reunião? Quais os impedimentos que bloqueiam esse trabalho?;  
  - *Sprint* *Review Meeting*: é a cerimônia na qual as *user stories* desenvolvidas são apresentadas para o *product owner* e demais envolvidos, e o *product owner* analisa se o que foi desenvolvido pode ser considerado concluído. O resultado dessa reunião é a definição se temos um incremento do software aprovado ou não pelo *product owner*;  
  - *Sprint* *Retrospective*: é uma reunião aberta onde todos devem expressar suas opiniões a fim de implementar melhorias para as próximas *sprint*s. Os pontos levantados devem ir para um plano de ação com itens a serem melhorados.

DDD – Introdução a Domain Driven Design

Domain Driven Design significa Projeto Orientado a Domínio. Ele veio do título do livro escrito por Eric Evans, dono da DomainLanguage, uma empresa especializada em treinamento e consultoria para desenvolvimento de software. O livro de Evans é um grande catálogo de Padrões, baseados em experiências do autor ao longo de mais de 20 anos desenvolvendo software utilizando técnicas de Orientação a Objetos. O que seria um Padrão?

Um padrão é uma regra de três partes que expressa a relação entre um contexto (1), um problema (2) e uma solução (3).

DDD pode ser visto por alguns como a volta da orientação a objetos. É verdade que o livro é um chamado às boas práticas de programação que já existem desde a época remota do SmallTalk. Quando se fala em Orientação a Objetos pensa-se logo em classes, heranças, polimorfismo, encapsulamento. Mas a essência da Orientação a Objetos também tem coisas como:

Alinhamento do código com o negócio: o contato dos desenvolvedores com os especialistas do domínio é algo essencial quando se faz DDD (o pessoal de métodos ágeis já sabe disso faz tempo);

* + - * Favorecer reutilização: os blocos de construção, que veremos adiante, facilitam aproveitar um mesmo conceito de domínio ou um mesmo código em vários lugares;
      * Mínimo de acoplamento: Com um modelo bem feito, organizado, as várias partes de um sistema interagem sem que haja muita dependência entre módulos ou classes de objetos de conceitos distintos;
* Independência da Tecnologia: DDD não foca em tecnologia, mas sim em entender as regras de negócio e como elas devem estar refletidas no código e no modelo de domínio. Não que a tecnologia usada não seja importante, mas essa não é uma preocupação de DDD.

Todas essas coisas são bem exemplificadas e mostradas na forma de vários padrões em DDD. Mas o livro também mostra muitos padrões que não dizem respeito a código ou modelagem. Aparecem coisas que estão mais ligadas a processos (como Integração Contínua) ou a formas de relacionamento entre times que fazem parte do desenvolvimento de um sistema complexo.

Test-Driven Development (TDD)

Test-Driven Development (TDD), sem dúvida, tornou-se uma das práticas mais populares entre desenvolvedores de software. A ideia é bem simples: escreva seus testes antes mesmo de escrever o código de produção. Mas por quê a ideia parece tão boa? Ao escrever os testes antes, o desenvolvedor garante que boa parte (ou talvez todo) do seu sistema tem um teste que garante o seu funcionamento. Além disso, muitos desenvolvedores também afirmam que os testes os guiam no projeto de classes do sistema.

O que eu ganho com a prática?

A prática de TDD agrega muitos benefícios ao processo de desenvolvimento. O primeiro deles, e mais claro, são os benefícios na qualidade externa do produto. Todos já sofremos os problemas de uma nova versão do produto que traz novas funcionalidades, mas faz as anteriores pararem de funcionar. A bateria de testes automatizados gerados pela prática dão mais segurança ao desenvolvedor na hora de mudanças.

Os testes automatizados, que rodam em questão de segundos, são executados o tempo todo pelo desenvolvedor. Isso quer dizer que podemos executá-los o dia todo, muitas vezes por dia. Algo que sabemos ser impossível com testes manuais. Caso algo pare de funcionar, o desenvolvedor é rapidamente notificado, e consegue corrigir o problema antes de mandar a versão para o cliente. E todos nós sabemos o quanto é bom não estressar o usuário final com bugs, não é verdade?

Além disso, muitos autores populares da área afirmam que, caso o desenvolvedor saiba ler o código de testes com atenção, esse mesmo código pode dar informações importantes sobre a qualidade do código que está sendo produzido. Dizemos que a prática de TDD ajuda o desenvolvedor a escrever código de produção de qualidade. É difícil explicar esses efeitos em poucas palavras, mas a ideia geral é que se está difícil escrever um teste automatizado, é porque provavelmente o código de produção está complicado. Essa é uma ótima dica para o desenvolvedor.

Perceba então que o uso da prática de TDD ajuda a equipe a garantir que os requisitos funcionam como esperado, e também auxilia o desenvolvedor a perceber problemas de código em suas implementações. Dois coelhos em uma cajadada só.

## 4 – Frameworks agile

### 4.1 - XP - Extreme Programming

Definição

O XP é um método de desenvolvimento de software, leve, não é prescritivo, e procura fundamentar as suas práticas por um conjunto de valores que serão vistos posteriormente no artigo. O XP, diferentemente do que muito pensam, também pode ser adotar por desenvolvedores médios e não apenas por desenvolvedores experientes.

Valores

Os cinco valores de XP são comunicação, simplicidade, feedback, coragem e respeito e são descritos em mais detalhes abaixo.

Comunicação

Desenvolvimento de software é inerentemente um esporte de equipe que se baseia em comunicação para transferir conhecimento de um membro da equipe para todos os outros na equipe. XP sublinha a importância do tipo apropriado de comunicação - cara a cara discussão com a ajuda de um quadro branco ou outro mecanismo de desenho.

Simplicidade

Simplicidade significa “o que é a coisa mais simples que vai funcionar?” O objetivo deste é para evitar o desperdício e fazer apenas coisas absolutamente necessárias, tais como manter o design do sistema tão simples quanto possível, de modo que é mais fácil de manter, apoio e rever. Simplicidade também significa endereço apenas os requisitos que você conhece; não tentar prever o futuro.

Comentários

Através do feedback constante sobre os seus esforços anteriores, as equipes podem identificar áreas de melhoria e rever suas práticas. Feedback também suporta design simples. Sua equipe constrói alguma coisa, reúne feedback sobre o seu projeto e implementação, e depois ajustar o seu produto vai para a frente.

Coragem

Kent Beck definido coragem como “ação efetiva em face do medo” (Extreme Programming Explained P. 20). Esta definição mostra uma preferência por ação com base em outros princípios para que os resultados não são prejudiciais para a equipe. Você precisa de coragem para levantar questões organizacionais que reduzem a eficácia da sua equipe. Você precisa de coragem para parar de fazer algo que não funciona e tentar outra coisa. Você precisa de coragem para aceitar e agir em feedback, mesmo quando é difícil de aceitar.

Respeito

Os membros de sua equipe precisa respeitar uns aos outros, a fim de se comunicar uns com os outros, fornecer e aceitar feedback que honra o seu relacionamento, e trabalhar em conjunto para identificar projetos e soluções simples.

Práticas

O núcleo do XP é o conjunto interligado de práticas de desenvolvimento de software listados abaixo. Embora seja possível fazer essas práticas em isolamento, muitas equipes têm encontrado algumas práticas reforçam os outros e deve ser feito em conjunto para eliminar totalmente os riscos que muitas vezes enfrentam no desenvolvimento de software. As Práticas XP mudaram um pouco desde que eles foram inicialmente introduzidos. As doze práticas estão listados abaixo.

* O Jogo do Planejamento
* Pequenos Releases
* Metáfora
* Design simples
* Prova
* Refatoração
* A programação em pares
* Propriedade coletiva
* Integração contínua
* Semana de 40 horas
* No local do cliente
* Padrões de codificação

Abaixo estão as descrições das práticas conforme descrito na segunda edição do Extreme Programming Explained aceitar a mudança. Estas descrições incluem refinamentos com base em experiências de muitos que praticam programação extrema e refletem um conjunto mais prático de práticas.

Sente junto

Como a comunicação é um dos cinco valores de XP, ea maioria das pessoas concorda que conversa cara a cara é a melhor forma de comunicação, ter sua equipe de sentar-se juntos no mesmo espaço, sem barreiras à comunicação, tais como as paredes dos cubículos.

Toda a Equipe

Um grupo funcional transversal de pessoas com os papéis necessários para um produto formar uma única equipe. Isso significa que as pessoas com uma necessidade, bem como todas as pessoas que desempenham uma parte em satisfazer essa necessidade todos trabalhar juntos em uma base diária para conseguir um resultado específico.

Workspace informativo

Configure o seu espaço de equipa para facilitar a comunicação face a face, permitir que as pessoas têm um pouco de privacidade quando necessário, e fazer o trabalho da equipe transparentes uns aos outros e às partes interessadas fora da equipe. Utilizar informações Radiadores para se comunicar ativamente informações up-to-date.

Trabalho energizado

Você é mais eficaz no desenvolvimento de software e todo o trabalho de conhecimento, quando você está focado e livre de distrações. Trabalho energizado significa tomar medidas para se certificar de que você é capaz fisicamente e mentalmente para entrar em um estado focado. Isto significa que não sobrecarregar-se (ou deixar que os outros sobrecarregar você). Ele também significa manter saudável, e mostrar respeito aos seus companheiros de equipe para mantê-los saudáveis.

A programação em pares

Pair Programming significa todos os softwares de produção é desenvolvido por duas pessoas sentadas na mesma máquina. A ideia por trás desta prática é que dois cérebros e quatro olhos são melhores do que um cérebro e olhos. Você efetivamente obter uma revisão de código contínuo e resposta mais rápida aos problemas irritantes que pode parar uma pessoa morta em suas trilhas.

As equipes que usaram a programação em pares descobriram que ele melhora a qualidade e não realmente ter o dobro do tempo, porque eles são capazes de trabalhar com os problemas mais rápido e eles ficam mais focados na tarefa em mãos, criando assim menos código para realizar a mesma coisa.

Histórias

Descrever o que o produto deve fazer em termos significativos para clientes e usuários. Estas histórias são destinados a ser breves descrições de usuários coisas quero ser capaz de fazer com o produto que pode ser usado para o planejamento e servir como lembretes para conversas mais detalhadas quando a equipe fica em torno de perceber que a história particular.

Ciclo semanal

O ciclo semanal é sinônimo de uma interação. No caso do XP, a equipe se reúne no primeiro dia da semana para refletir sobre os progressos até à data, o cliente escolhe as histórias que gostaria entregues nessa semana, ea equipe determina como eles vão abordar essas histórias. A meta até o final da semana é ter correndo características testadas que realizam as histórias selecionadas.

A intenção por trás o período de tempo de entrega encaixotado é produzir algo para mostrar para o cliente para o gabarito.

Ciclo Quarterly

O ciclo trimestral é sinônimo de autorização. O objetivo é manter o trabalho detalhado de cada ciclo semanal no contexto do projeto global. O cliente estabelece o plano geral para a equipe em termos de características desejadas dentro de um quarto particular, que fornece a equipe com uma vista para a floresta, enquanto eles estão nas árvores, e também ajuda o cliente a trabalhar com outras partes interessadas, que podem precisa de alguma ideia de quando os recursos estarão disponíveis.

Lembre-se quando planear um ciclo trimestral as informações sobre qualquer história particular está em um nível relativamente alto, a ordem de entrega história dentro de uma ciclo trimestral pode mudar e as histórias incluídas no ciclo trimestral pode mudar. Se você é capaz de rever o plano em uma base semanal, após cada ciclo semanal, você pode manter todos informados, logo que essas alterações se tornam aparentes para manter surpresas ao mínimo.

Negligente

A ideia por trás folga em termos XP é adicionar algumas baixas tarefas prioritárias ou histórias em seus ciclos semanais e trimestrais que podem ser descartados se a equipe fica atrás em tarefas mais importantes ou histórias. Dito de outra forma, explicar a variabilidade inerente nas estimativas para se certificar de que você deixar-se uma boa chance de satisfazer as suas previsões.

Desenvolver dez minutos

O objetivo com a construção de dez minutos é construir automaticamente todo o sistema e executar todos os testes em dez minutos. Os fundadores da XP sugeriu um quadro tempo de 10 minutos, porque se uma equipe tem uma construção que leva mais tempo do que isso, é menos provável de ser executado em uma base frequente, introduzindo assim mais tempo entre erros.

Esta prática incentiva sua equipe para automatizar o processo de compilação para que você são mais propensos a fazê-lo em uma base regular e usar esse processo de compilação automatizada para executar todos os testes.

Esta prática suporta a prática de integração contínua e é apoiado pela prática de teste primeiro desenvolvimento.

Integração contínua

Integração contínua é uma prática onde as mudanças de código são imediatamente testado quando eles são adicionados a uma base de código maior. O benefício desta prática é que você pode pegar e corrigir problemas de integração mais cedo. A maioria das equipes temer o passo de integração de código por causa da descoberta inerente de conflitos e problemas que resultam. A maioria das equipes adotar a abordagem “Se dói, evitá-lo o maior tempo possível”.

Praticantes de XP sugerir “se dói, fazer isso mais vezes”.

O raciocínio por trás dessa abordagem é que, se você tiver problemas cada vez que você integrar o código, e isso leva um tempo para descobrir onde os problemas são, talvez você deve integrar mais frequentemente de modo que se houver problemas, eles são muito mais fáceis de encontrar porque não são menos alteraes incorporadas na construção.

Esta prática requer alguma disciplina extra e é altamente dependente Ten Desenvolver Minuto e teste primeiro desenvolvimento.

Test-Primeira Programação

Em vez de seguir o caminho normal:

Desenvolver código -> Testes de escrita -> executar testes

A prática de Test-primeira programação segue o caminho de:

Escrever não teste automatizado -> Executar teste de falha -> desenvolver o código para fazer passagem de teste -> teste executar -> repeat

Tal como acontece com Integração Contínua, Test-primeira programação reduz o ciclo de feedback para os desenvolvedores a identificar e resolver problemas, diminuindo assim o número de bugs que são introduzidas em produção.

Projeto incremental

A prática da Incremental projeto sugere que você faça um pouco de trabalho na frente para entender a perspectiva largura-wise adequada do projeto do sistema, e em seguida, mergulhar nos detalhes de um aspecto particular do que o design quando você entregar características específicas. Essa abordagem reduz o custo de mudanças e permite-lhe tomar decisões de design, quando necessário, com base na informação mais recente disponível.

A prática de refatoração foi originalmente colocados entre o núcleo 12, mas foi incorporada à prática da Incremental projeto. Refatoração é uma excelente prática de usar para manter o design simples, e um dos usos mais recomendadas de refatoração é remover a duplicação de processos.

Roles

Embora Extreme Programming especifica práticas particulares para a sua equipe a seguir, ele realmente não estabelecer papéis específicos para as pessoas em sua equipe.

Dependendo da fonte que você lê, ou não há orientação, ou há uma descrição de como os papéis normalmente encontrados em projetos mais tradicionais se comportam em projetos de Extreme Programming. Aqui estão quatro papéis mais comuns associados com Extreme Programming:

O cliente

O papel do cliente é responsável por fazer todas as decisões de negócios sobre o projeto, incluindo:

Qual deve ser o sistema de fazer (o que recursos estão incluídos e o que eles realizar)?

Como sabemos quando o sistema é feito (quais são os nossos critérios de aceitação)?

Quanto temos que gastar (o que é o financiamento disponível, o que é o caso de negócio)?

O que devemos fazer a seguir (em que ordem é que vamos entregar esses recursos)?

O Cliente XP é esperado para estar ativamente engajados no projeto e, idealmente, passa a fazer parte da equipe.

O Cliente XP é assumido como sendo uma única pessoa, no entanto a experiência tem demonstrado que uma pessoa não pode fornecer adequadamente toda a informação relacionada com negócios sobre um projeto. Sua equipe precisa ter certeza de que você obter uma imagem completa da perspectiva de negócios, mas têm alguns meios de lidar com conflitos em que a informação de modo que você pode obter uma direção clara.

O desenvolvedor (The Developer)

Porque XP não tem muita necessidade de definição de função, todos na equipe (com exceção do cliente e um par de papéis secundários listados abaixo) é rotulado como um desenvolvedor. Os desenvolvedores são responsáveis ​​por realizar as histórias identificadas pelo Cliente. Porque diferentes projetos exigem uma combinação diferente de habilidades, e porque o método XP conta com uma Equipe Multidisciplinar proporcionando a combinação adequada de habilidades, os criadores do XP não sentia necessidade de mais definição de função.

O Rastreador (The Tracker)

Algumas equipes podem ter um rastreador como parte de sua equipe. Isso é muitas vezes um dos desenvolvedores que passa parte do seu tempo a cada semana preenchendo esse papel extra. O objetivo principal deste papel é o de manter o controle de métricas relevantes que a equipe sente necessário acompanhar o seu progresso e identificar áreas de melhoria. Métricas-chave que sua equipe pode rastrear incluem velocidade, razões para as alterações a velocidade, quantidade de horas extras, e passagem e testes fracassados.

Este não é um papel necessário para a sua equipe, e geralmente só é estabelecida se o seu time determina uma verdadeira necessidade de manter o controle de várias métricas.

O Treinador (The Coach)

Se sua equipe está apenas começando aplicação XP, você pode achar que é útil para incluir um treinador em sua equipe. Isso geralmente é um consultor externo ou alguém de outro lugar na sua organização que tenha usado XP antes e está incluído na sua equipe para ajudar a orientar os outros membros da equipe sobre as práticas XP e para ajudar sua equipe a manter a sua autodisciplina.

O principal valor do treinador é que eles passaram por isso antes e pode ajudar sua equipe a evitar erros que a maioria das novas equipes fazem.

### 4.2 - FDD - Feature Driven Development

Entre as metodologias ágeis que existiam antes do manifesto ágil, o FDD “Feature Driven Developement” é uma delas. Concebido originalmente por Jeff de Luca, o FDD surgiu em Singapura, entre os anos de 1997 e 1999 com base no método Coad (Criado por Peter Coad – 1980/1990) e nos processos interativos e lean já utilizados por Jeff de Luca.

O FDD busca o desenvolvimento por funcionalidade, ou seja, por um requisito funcional do sistema. É pratico para o trabalho com projetos iniciais ou projetos com codificações existentes. Apesar de ter algumas diferenças entre o FDD e o XP, é possível utilizar as melhores práticas de cada metodologia. O FDD atua muito bem em conjunto com o Scrum, pois o Scrum atua no foco do gerenciamento do projeto e o FDD atua no processo de desenvolvimento.

O FDD possui cinco processos básicos.

* Desenvolvimento de modelo abrangente (Análise orientada por objetos);
* Construção de lista de funcionalidades (Decomposição funcional);
* Planejar por funcionalidade (Planejamento incremental);
* Detalhe por funcionalidade (Desenho orientado a objetos);
* Construção por funcionalidade (Programação e teste orientado a objetos).

Assim como acontece na metodologia XP, o FDD faz uso de teste de software. Desta forma é possível utilizar TDD, aliás, é indicada a utilização deste para manter a qualidade do software.

O FDD, assim como a teoria de sistemas afirma, entende que a soma das partes é maior do que o todo. Desta forma, apesar de criar um modelo abrangente baseado no todo que será desenvolvido, esta fase inicia-se com o detalhamento do domínio do negócio com divisão de áreas que serão modeladas. O modelo só está pronto quando todos da equipe estiverem de acordo, o planejamento é feito com base na lista de funcionalidades. Se fossemos trabalhar com FDD em conjunto com o Scrum, a lista de funcionalidades seria utilizada no backlog de produtos, como histórias de usuários a serem desenvolvidas.

Com base na lista de funcionalidades, deve-se planejar por funcionalidade, mas este planejamento é incremental. Isto em conjunto com o Scrum, deve ser analisado como etapa de desenvolvimento do incremento, então este planejamento é feito com base no que será desenvolvido naquele incremento.

Vamos novamente ao Scrum, separando o que será feito na Sprint. Colocamos no backlog da Sprint. O que está no backlog da sprint são funcionalidades que serão desenvolvidas durante a sprint (que pode ser de duas a quatro semanas), estas tarefas são então planejadas com maior rigor, neste ponto, temos então o planejamento incremental, ou seja, planejamos apenas o que vamos desenvolver. Nesta etapa os envolvidos no processo resumem-se apenas à equipe, ou seja, os desenvolvedores, analistas, testadores, etc., que vão atuar no processo. Eles devem selecionar os itens com base no tempo que eles possuem e nas qualificações atuais da equipe.

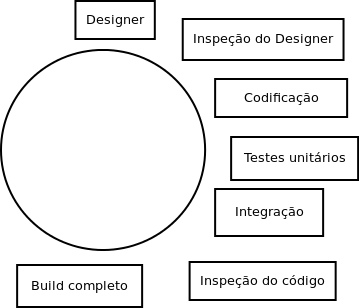
Após o planejamento, é feito o detalhamento, nesta fase é de extrema importância que o desenho esteja de acordo com o que o cliente deseja, então é mantido contato constante com o cliente, como em todas as metodologias ágeis.

Esta documentação é a base para o desenvolvimento, não deve-se perder tempo com documentação que não será utilizada, mas é necessário documentar.

Posteriormente inicia-se a fase de desenvolvimento, esta fase é a etapa de desenvolvimento e teste real.

O desenvolvimento também é incremental, e indica-se a utilização de testes do início ao fim do processo, além da utilização de integração continua.

Um ponto que diverge do XP é que no FDD é incentivado o desenvolvedor como único responsável pelo módulo que este desenvolve, já no XP, o código é comunitário.

  
Figura 4: Integração contínua

A utilização da integração continua permite que a equipe esteja em contato constante com o cliente, tornando o processo ágil e com entregas constantes.

Como cada fase apresentada acima é específica e curta, e as fases se completam, são necessários relatórios para manter o controle, para analisar a quantidade de recursos que estão sendo desenvolvidos, semelhante ao burndow e o burnup do Scrum.

Segundo a metodologia, o percentual de tempo gasto em cada etapa é:

* Levantamento do domínio da aplicação = 1%;
* Projeto = 40%;
* Inspeção do projeto = 3%;
* Desenvolvimento = 45%;
* Inspeção do código = 10%;
* Integração = 1%.

Além disso, o FDD possui as chamadas melhores práticas que indicam boas práticas ao desenvolver com o FDD, são elas:

* Modelagem Orientada a Objetos do Domínio;
* Desenvolvimento por funcionalidade;
* Classe proprietária, ou seja, a unidade é feita individualmente, evitando-se assim conflitos na equipe;
* Equipes de recursos: são equipes pequenas, destinadas a desenvolver recursos necessários ao projeto, de forma secundária;
* Inspeção é realizada constantemente para garantir a boa qualidade do código e do projeto;
* Gerenciamento de configuração;
* Integração contínua para demonstrar constantemente as funcionalidades ao cliente e;
* Visibilidade de progressos e resultados.

Na equipe do FDD temos:

Para melhor definir as atribuições de cada integrante de um projeto, o FDD descreve seis papéis principais(gerente de projetos, arquiteto chefe, gerente de desenvolvimento, programadores chefes, donos de classes, especialistas de domínio)seis papéis de apoio (gerente de domínio, gerente de versão, “guru” da linguagem, engenheiro construtor, criador de ferramentas, administrador de sistemas) e três papéis adicionais(testador, implantador e escritor técnico).

***Gerente de Projetos***

É ele o responsável por gerenciar o projeto, seus recursos, seus prazos, as tecnologias utilizadas. É ele também que deve sofrer todas as interações externas (cobranças do cliente ou da diretoria, por exemplo) blindando todo o restante da equipe envolvida no desenvolvimento do projeto. Basicamente, pode-se dizer que ele é o responsável por permitir que o trabalho de todos outros envolvidos flua sem maiores problemas e que o projeto seja entregue corretamente.

***Arquiteto Chefe***

O arquiteto chefe é o responsável pelo projeto geral do sistema, ou seja, ele que dará a palavra final sempre que houver qualquer discussão acerca dos modelos a serem utilizados para o desenvolvimento, servindo muitas vezes como um intermediador entre os programadores chefes.

***Gerente de Desenvolvimento***

Cabe ao gerente de desenvolvimento fazer com que o desenvolvimento em si tenha um andamento normal, sem interrupções. A pessoa que estiver nesse papel deve ter boas habilidades de desenvolvimento e deve resolver qualquer conflito de recursos que surja entre os programadores chefes no dia-a-dia.

***Programadores Chefes***

São desenvolvedores mais experientes que já participaram de alguns ciclos completos de desenvolvimento de software. Eles lideram times pequenos de desenvolvedores e atuam na análise e projeto de atividades mais complexas. Cabe a eles também entenderem-se com os outros programadores chefes em conflitos técnicos ou por recursos.

***Donos de Classes***

Dentro dos times liderados pelos programadores chefes, são as figuras mais importantes (após, é claro, o próprio programador chefe), pois são responsáveis por projetar, codificar, testar, documentar e consolidar tudo que for feito em suas determinadas classes.

***Especialistas de Domínio***

Podendo ser representado por clientes, analistas de negócio, ou por uma mistura deles, o especialista de domínio é, como o nome sugere, alguém que possui muitos conhecimentos no negócio para o qual o sistema será desenvolvido ou sobre algum negócio com o qual o sistema deverá se relacionar.

***Gerente de versão***

Basicamente, o gerente da versão é o responsável por uma versão, ou iteração, que está sendo desenvolvido, monitorando o trabalho dos programadores chefes e verificando se será entregue tudo o que foi planejado. Ele trabalha em conjunto com o gerente de projetos, relatando a ele como está o desenvolvimento da versão.

***“Guru” da linguagem***

O “guru” da linguagem, como o nome sugere, é alguém que saiba muito ou tudo sobre uma linguagem ou tecnologia específica que será utilizada no projeto, esse papel é especialmente importante nos casos em que será utilizada uma linguagem ou tecnologia pela primeira vez, tornando-se não tão importante nos casos em que a maior parte da equipe (ou ela toda) já conhece a tecnologia em questão.

***Engenheiro Construtor***

O engenheiro construtor deve preparar tudo o que for necessário para que a próxima versão possa funcionar sem problemas (scripts preparatórios, geração de documentação ou relatórios) além de garantir que a versão do sistema esteja íntegra (controlar a versão para que nada seja perdido durante as consolidações e que as novas alterações sejam consolidadas corretamente).

***Criador de ferramentas***

Cria sistemas e ferramentas para auxiliarem no desenvolvimento do projeto, seja para o próprio desenvolvimento ou para a fase de testes ou conversão de dados.

***Administrador do Sistema***

Configura e gerencia o ambiente de rede dos desenvolvedores e de testes, garantindo que o trabalho deles possa ocorrer sem problemas de infraestrutura.

***Testador***

O testador é o responsável por verificar se o que foi desenvolvido está de acordo com o que foi solicitado e se atende plenamente a necessidade do cliente (assim como nas metodologias tradicionais e nas outras ágeis).

***Implantador***

Faz a conversão necessária de dados para que o banco de dados antigo fique compatível com o novo sistema. Enfim, faz a atualização do ambiente para que o novo sistema possa rodar sem maiores problemas.

***Escritor técnico***

Escreve e prepara toda a documentação impressa e digital para o usuário.

### 4.3 - Kanban

O método Kanban foi concebido como um caminho alternativo à agilidade organizacional. Ao invés de grandes mudanças, o Kanban propõe uma abordagem evolucionária, onde pequenas melhorias são integradas na prestação de um serviço ou desenvolvimento de um produto.

Nascido dentro do Sistema Toyota de Produção (TPS), o kanban (com k minúsculo) era um cartão utilizado para sinalizar conclusão de etapas do processo e tornar o fluxo puxado. Mais tarde, David J. Anderson criou o método Kanban (com K maiúsculo) e popularizou o seu uso no desenvolvimento de produtos e serviços. Neste guia detalharemos o funcionamento deste segundo.

* Entendendo um sistema Kanban
* Sistema empurrado
* Sistema puxado
* Os princípios básicos do método Kanban
* As 6 práticas gerais de um sistema Kanban
* Quando usar o método Kanban?
* Vantagens sobre o Scrum
* Como implementar um sistema Kanban
* Elementos de um sistema Kanban
* Métricas de um sistema Kanban
* Método Kanban & Autogestão (O2)
* E tem muito mais que ainda não está aqui

Um sistema Kanban é composto por um fluxo de valor, onde unidades de trabalho trafegam da esquerda para a direita. Cada etapa do processo adiciona mais valor ao item, sendo que quando ele chega ao final do processo, ele está “concluído”. Esse fluxo de valor pode ser o desenvolvimento de um software, a prestação de um serviço (como atendimento ao cliente) ou até a criação de um produto físico.

**OS PRINCÍPIOS BÁSICOS DO MÉTODO KANBAN**

Em seu livro Kanban: successful evolutionary change for your technology business, David J. Anderson descreve os princípios básicos do método Kanban:

Comece com o que você tem hoje. O método Kanban propõe uma abordagem evolucionária e incremental. Por mais que você esteja muito insatisfeito com a forma como as coisas são feitas no processo atual, não mude tudo logo no início. Fazer grandes mudanças pode aumentar a resistência das pessoas, além de ser muito arriscado para a organização.

Busque mudanças incrementais e evolucionárias. Depois de partir do seu processo atual, busque pequenas mudanças. Formule hipóteses com base na sua observação do comportamento do sistema. Seja curioso e faça experimentos.

Vamos supor que o seu sistema possui uma coluna onde um grande documento é produzido que detalha o que deve ser feito. Talvez você desconfie que o nível de detalhe seja grande demais e que isso é um provável desperdício. Formule a hipótese: Se nós simplificarmos o documento, teremos uma redução do Lead Time do processo. Você pode estar certo ou não. Faça um experimento, meça os resultados e compare.

Respeite o processo atual, papéis, responsabilidades e títulos. É provável que na organização em que você está implementando o método Kanban existam cargos e autoridades definidas. Talvez essa estrutura esteja atrapalhando o fluxo, na sua visão. Mas ao mesmo tempo, é também muito provável que boa parte dos processos atuais simplesmente funcione. Por isso é importante respeitar o que já está posto e perseguir a melhoria contínua a partir disso.

**AS 6 PRÁTICAS GERAIS DE UM SISTEMA KANBAN**

David J. Anderson descreve 6 práticas gerais das implementações de sucesso do método Kanban. Podemos dizer que sem elas, você ainda não está fazendo “Kanban”. Por isso que dissemos que o quadro inicial (que não tinha limites de trabalho em progresso) ainda não poderia ser considerado um sistema Kanban: ele falhava em uma das 6 práticas gerais.

As 6 práticas são:

Visualize o fluxo de trabalho. Você não pode gerenciar o que não pode ver! E se tratando de trabalho do conhecimento, esta afirmação é ainda mais verdadeira. Na Toyota, o estoque excessivo de carros poderia ser facilmente visualizado, já que um carro é um produto físico. Mas e todas as funcionalidades de um sistema que foram começadas, mas não terminadas? Ou todos os tickets de atendimento que foram abertos, lidos, mas não respondidos? O trabalho do conhecimento é intangível. É por isso que precisamos estabelecer formas tangíveis de visualizar o fluxo e as unidades de trabalho.

Limite o trabalho em progresso. Se você não estabelecer limites para o trabalho em andamento, é muito provável que você ainda esteja operando um sistema empurrado. Em geral, todas as etapas em um fluxo são limitadas, com exceção da última. Estabelecer limites vai impedir o acúmulo de estoque e também vai manter o Lead Time estável. Em geral, quanto mais itens houverem em andamento, maior é o tempo de entrega.

Gerencie o fluxo. No Kanban, focamos os esforços na otimização do sistema como um todo. A gestão tradicional empurrada foca em manter todas as pessoas ocupadas e gerenciar o trabalho delas. Isso não é feito no método Kanban. Se tentarmos otimizar uma parte do sistema (o trabalho de um desenvolvedor), isso provavelmente levará a uma situação sub-ótima globalmente. Esse fenômeno chama-se otimização local. Leia mais sobre como isso acontece com as metas.

Torne as políticas de processo explícitas. Ao trabalhar com um fluxo Kanban, você vai perceber que muitas regras de processos, papéis e definições não são claras para os participantes do fluxo. Vai surgir muita confusão sobre o significado de uma determinada coluna do fluxo, por exemplo. É importante que você aproveite essas oportunidades para esclarecer as políticas do processo e torná-las explícitas. Isso significa registrá-las em algum local acessível para todos. Parte importante de jogar o jogo é conhecer as regras.

Implemente ciclos de feedback. Nenhum processo pode evoluir sem ciclos de feedback. David J. Anderson propõe alguns rituais específicos para retroalimentar o sistema e permitir com que os participantes se adaptem comparando a situação atual do processo com as expectativas das partes interessadas.

Melhore colaborativamente. No final, as limitações de trabalho em progresso irão introduzir desconfortos e iniciar conversas sobre o fluxo. O que se busca em um sistema Kanban é um fluxo suave e constante. No entanto, não é isso que vai acontecer no início. É importante estabelecer um processo de melhoria colaborativa, onde o grupo constrói um entendimento compartilhado sobre os problemas encontrados e a partir daí propõe experimentos para melhorar o fluxo.

**CLASSIFICAÇÃO DE ITENS E HIERARQUIA**

A classificação de itens vai de equipe para equipe, não existe uma regra que dite quais tipos de itens devem ser utilizados, eles são definidos de acordo o processo de trabalho da equipe. Podemos citar alguns exemplos de item, como: Defeito, tarefas, estória de usuário, recurso, casos de uso, sugestão de melhoria, enfim, é possível classificar o item em diferentes tipos de demanda. Fica a seu critério identificar os itens referentes ao seu contexto, porém, leve em conta itens que referencie bem a classificação de seu quadro. Muitas vezes encontramos no Kanban itens que são mais abrangentes, requerendo assim uma hierarquia de rompimento. Vou citar para você três tipos de hierarquias que são comumente utilizadas, são elas:

Épicos (Epics)

Um épico é uma grande estória. Quando uma solicitação é muito extensa necessita ser quebrada em partes menores (estórias). Isso ajuda a manter princípios ágeis como, por exemplo, entregar software funcionando com frequência.

Estória de usuário (User Story)

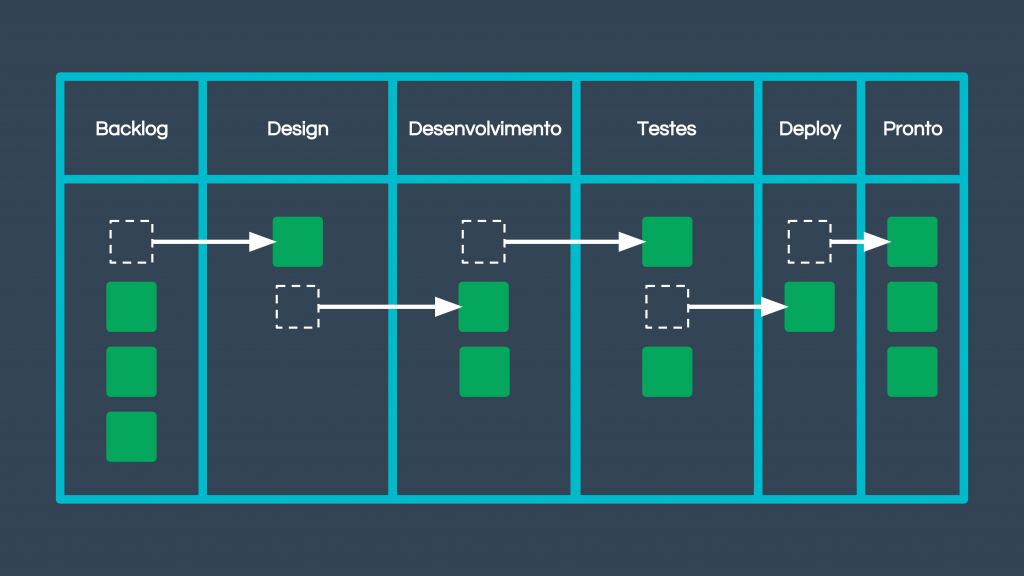
Uma estória de usuário pode ser caracterizada como uma curta e simples descrição de um recurso. Ela pode ser contada a partir da perspectiva da pessoa que deseja a nova capacidade, usualmente de um usuário ou cliente do sistema.

Tarefas (Tasks)

As tarefas são os elementos de uma estória. Por exemplo, se a estória de usuário é “Como usuário comum, eu quero me cadastrar na landing page através do formulário no topo, por que desejo receber o ebook prometido”, uma tarefa dessa estória poderia ser “Integrar API de email ao formulario”. Estórias de usuário que são muito pequenas, podem não ter a necessidade de ser divididas em tarefas.

**EVIDENCIANDO QUADROS KANBAN**

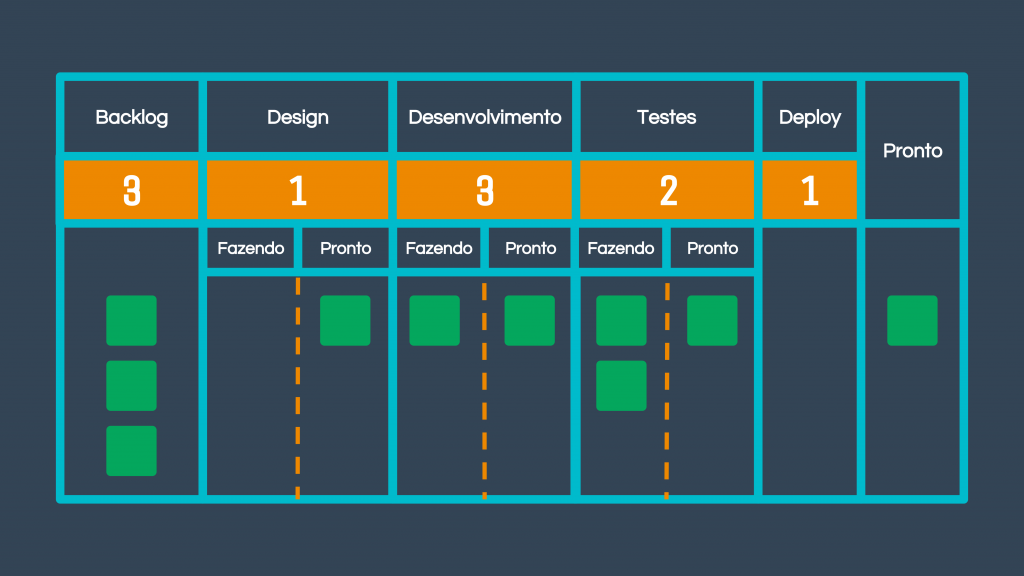
Vamos supor que fazemos parte de um time de desenvolvimento de um aplicativo. O nosso processo atual tem algumas etapas, que podemos mapear na forma de colunas “Backlog”, “Design”, “Desenvolvimento”, “Testes”, “Deploy” (implantação) e “Pronto”. Neste quadro Kanban, as unidades de trabalho representam coisas que geram algum benefício ao cliente/usuário final, como novas funcionalidades, correção de defeitos ou melhorias na interface do produto. O valor é gerado somente quando um item alcança a última etapa:



Apesar de conter etapas sequenciais e cartões, isso ainda não é um sistema Kanban. Vamos fazer mais duas mudanças para tornar o fluxo puxado:

* 1. Dividir as colunas intermediárias em dois estágios: “fazendo” e “pronto”;
  2. Adicionar limites de trabalho em progresso

Feitas essas mudanças, o nosso quadro ficaria assim:



Vamos analisar como isso funcionaria na prática. Supondo que você é o especialista em Design do time e que você está trabalhando em uma funcionalidade. Após você concluir a sua atividade, você movimenta o cartão para a etapa “Pronto” da coluna “Design”. Se um desenvolvedor (que atua na etapa “Desenvolvimento”) está livre, ele pode perceber que você sinalizou a conclusão da sua parte e então “puxar” o cartão para a etapa “Fazendo” da coluna “Desenvolvimento”.

Repare que o limite “1” da coluna “Design” é válido tanto para “Fazendo” quanto “Pronto”. No caso acima, temos um item que foi concluído pelo Designer, mas ainda não foi puxado. O que o Designer faz nesse caso? Nada. Ele apenas aguarda alguém puxar o item para Desenvolvimento. Nesse caso o slot seria liberado e ele poderia trabalhar em outro item.

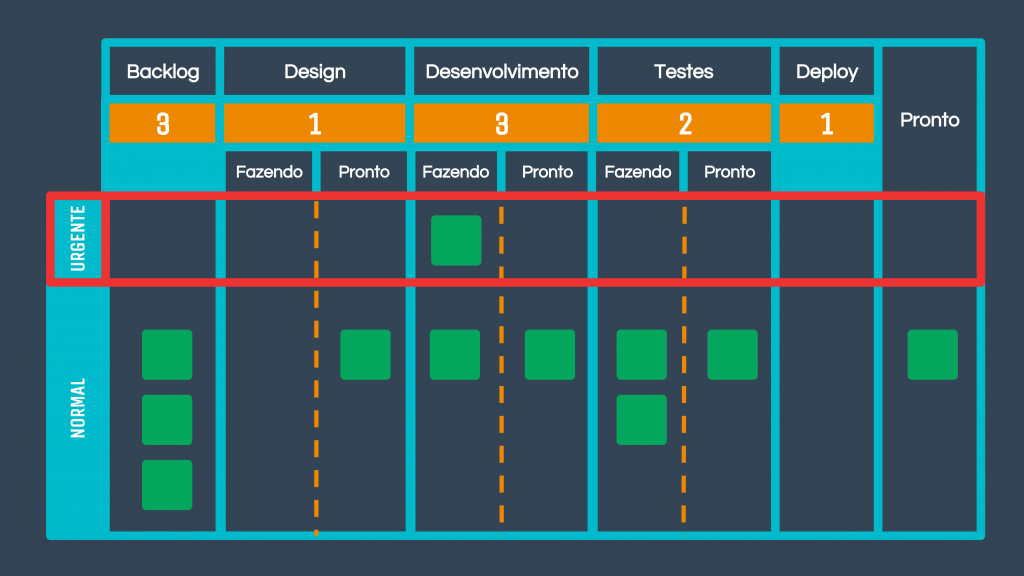
Claro que na prática se não houver slot disponível o membro do time não vai ficar simplesmente parado. Em um sistema Kanban, estimulamos as pessoas a olharem para o fluxo como um todo e buscarem formas de aumentar a vazão. Pode ser que o Designer possa ajudar alguém no Desenvolvimento. Pode ser que ele use esse tempo para pensar em melhorias. O que importa é não ficar simplesmente trabalhando na própria função, pois isso aumentaria o estoque e prejudicaria o fluxo.

Parece contra intuitivo, mas a teoria das filas comprova. A ociosidade no fluxo aumenta a vazão. Para entender os porquês, você vai precisar estudar um pouco de matemática (esse livro é um bom começo).

Como os itens são puxados apenas quando há capacidade disponível, raramente há sobrecarga. Isso é que o chamamos de sistema puxado. Ele é um contraste à forma tradicional de trabalhar, que chamamos de sistema empurrado.

Raias do quadro

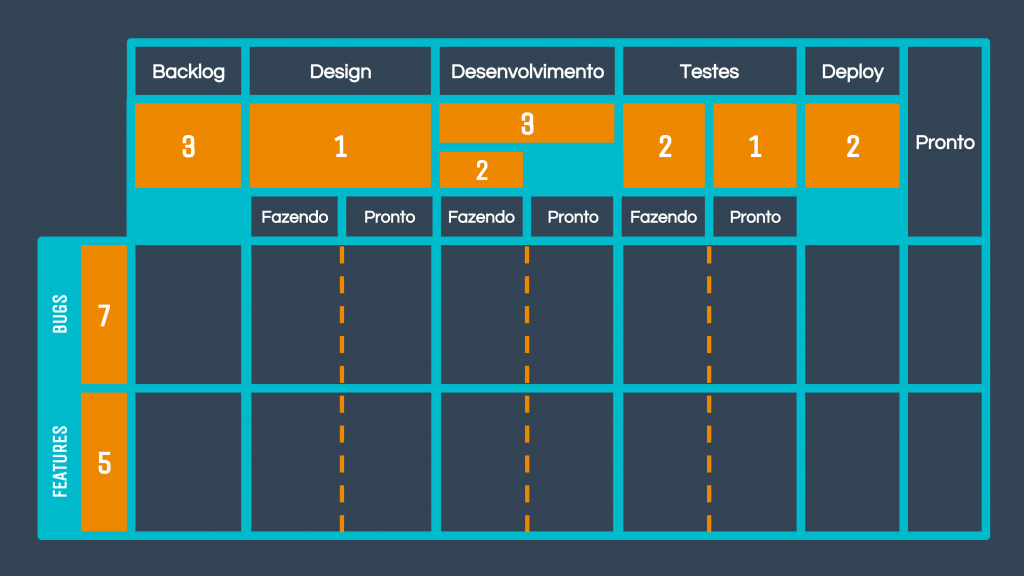
Além das colunas, que são divisões horizontais, muitos quadros possuem “raias”, que são o equivalente vertical. As raias costumam ser usadas para destacar visualmente uma classe de serviço ou para alocar capacidade por tipo de item.



No exemplo acima, utilizamos uma raia especial para destacar a classe de serviço urgente.

Limites de trabalho em progresso

Na introdução falamos sobre a importância dos limites de trabalho em progresso. Em termos de como montar o seu quadro, os limites podem ser estabelecidos por coluna, por subcoluna ou por raia. O quadro abaixo contém todos essas possibilidades.



A coluna “Design” possui um limite total de 1. Ou seja, podemos ter:

1 item em “Fazendo” e 0 itens em “Pronto”

0 itens em “Fazendo” e 1 item em “Pronto”

A coluna “Desenvolvimento” possui um limite total de 3 e um limite de 2 itens para a subcoluna “Fazendo”. A subcoluna “Pronto” não possui especificação de limite. Os seguintes cenários são permitidos:

2 itens em “Fazendo” e 1 item em “Pronto”

1 item em “Fazendo” e 2 itens em “Pronto”

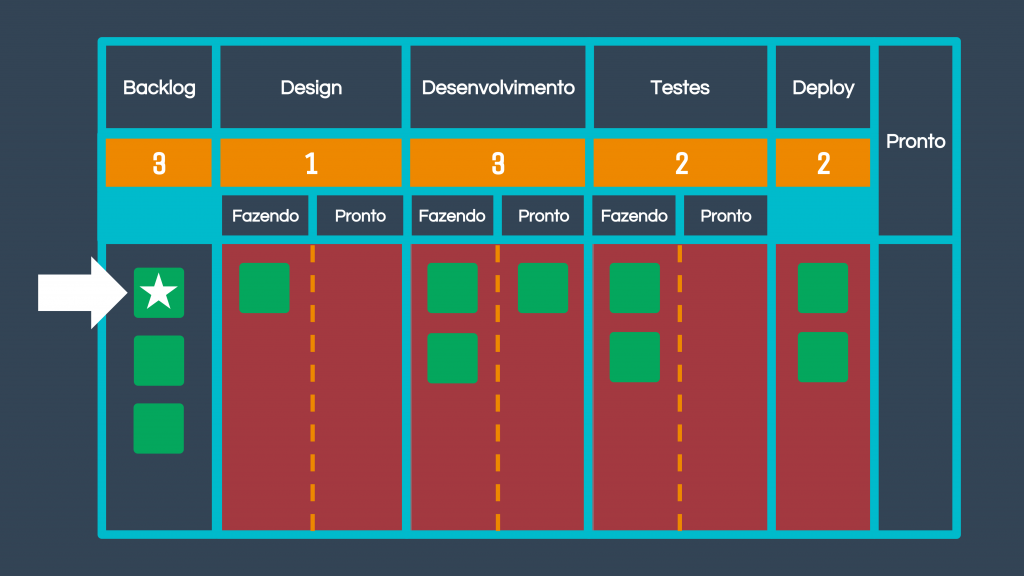
0 itens em “Fazendo” e 3 itens em “Pronto”

Nessa forma de representação, tanto os limites para a coluna quanto a subcoluna devem ser respeitados simultaneamente. Ou seja, não poderíamos ter 3 itens em “Fazendo” (o limite da subcoluna é 2) e nem 2 em “Fazendo” e 2 em “Pronto” (o limite total é 3).

Note também que existe uma alocação por capacidade (falaremos mais abaixo), com limites especificados para as raias “Bugs” e “Features”. Nesse caso os limites se aplicam a toda a linha. Ou seja, não poderemos ter mais do que 7 itens em toda a linha de bugs (independentemente da coluna) e 5 itens em features. A soma dos limites das linhas (5 + 7 = 12) deve ser igual à soma dos limites das colunas (3 + 1+ 3 + 2 + 1 + 2 = 12).

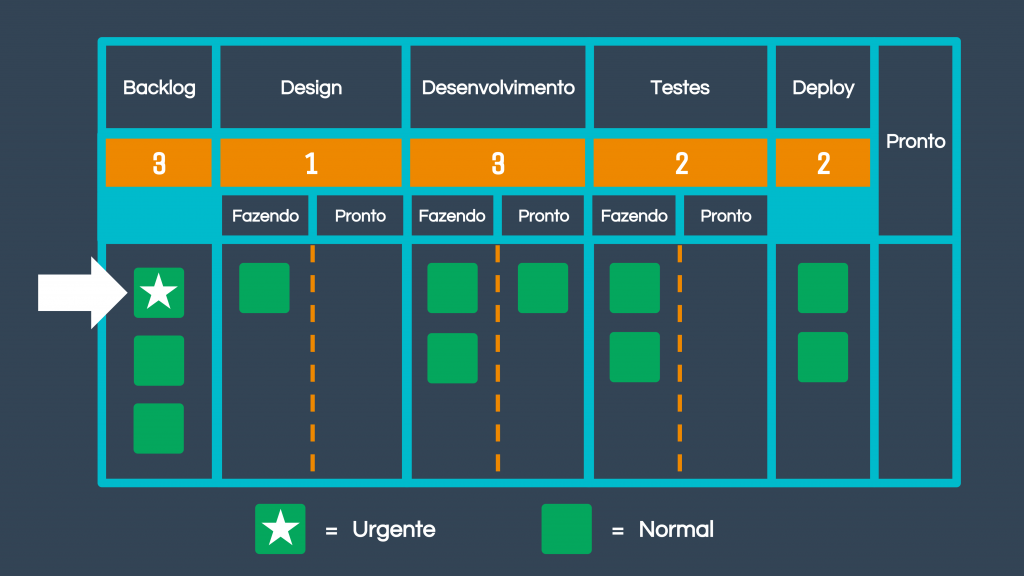
Classes de serviço

Imagine a seguinte situação: o seu sistema Kanban está operando em capacidade máxima (todas as colunas atingindo os limites de trabalho em progresso) e de repente aparece um novo item com alto custo de oportunidade. Pode ser um defeito crítico, que se não for corrigido a tempo pode causar um grande prejuízo à organização. Ou pode ser uma nova funcionalidade, que se lançada antes do Natal (data de entrega), pode trazer um aumento de 20% na receita.



Se o seu fluxo está cheio, o que você faz? Mesmo que você coloque ele em primeiro lugar na sua fila de entrada, é provável que ele demore muito tempo para ser concluído. Você também não pode simplesmente aumentar os limites, porque isso iria contra as políticas já definidas.

Itens como os descritos acima normalmente aparecem nas piores horas, por isso ter uma forma de responder rapidamente é importante. É justamente essa a função das classes de serviço. Com elas, podemos oferecer tratativas diferentes a itens com essas características.



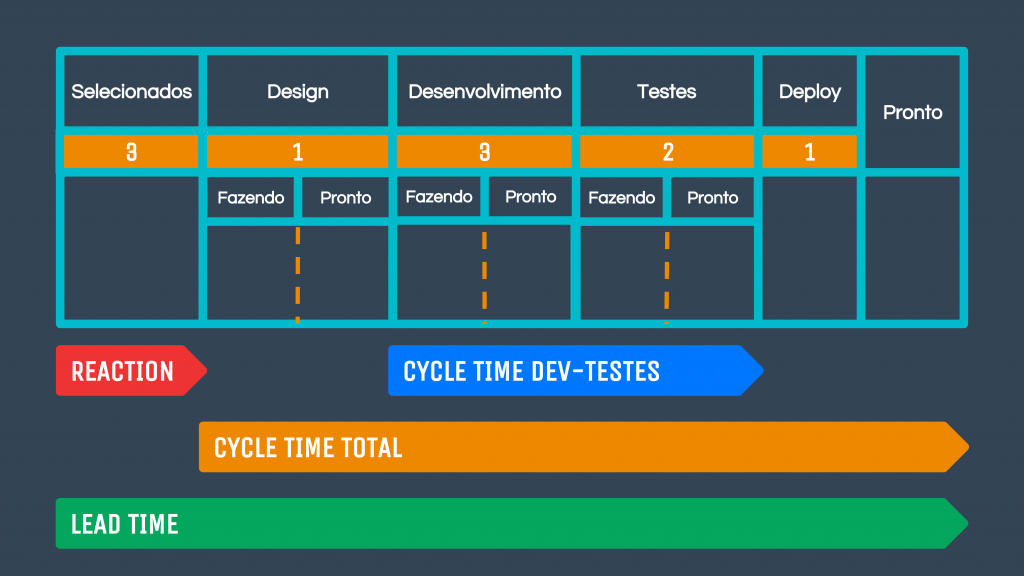
No quadro acima, temos uma marcação (a estrela) no item que define que ele possui a classe de serviço urgente. Como parte das políticas explícitas, poderíamos ter uma regra em que itens urgentes devem ser puxados primeiro e que podem exceder o limite de trabalho em progresso do sistema inteiro em 1. Poderíamos também incluir uma regra de que é possível ter no máximo 1 item urgente por vez, para não afetarmos muito o tempo de entrega dos outros itens.

Métricas de um sistema kanban

Na seção sobre as 6 práticas gerais do método Kanban, vimos que o foco deve ser em gerenciar o fluxo, não as pessoas. Só podemos melhorar continuamente se pudermos melhor observar o que acontece no processo atual. Para isso, utilizamos algumas métricas e diagramas. A seguir estão as mais comuns e importantes métricas de um sistema Kanban.

Cycle Time e Reaction Time

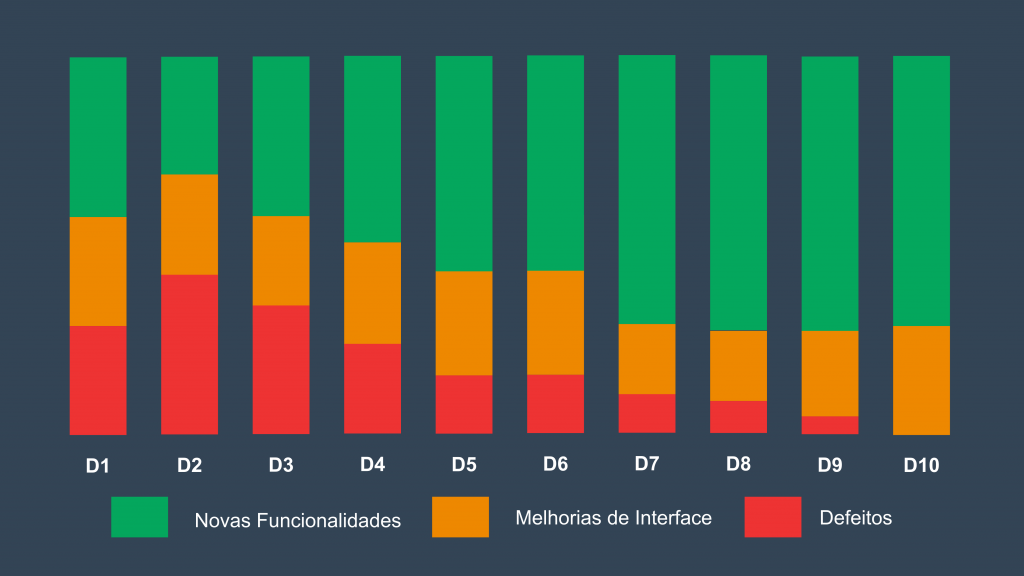
O tempo de reação e ciclo são complementos ao lead time. Perceba que no fluxo abaixo há uma fila de entrada chamada “Selecionados”. Ali são colocados um lote de itens a serem consumidos ao longo dos próximos dias. O tempo de reação é a diferença entre o momento que você começa a capturar o lead time (nesse fluxo, o momento em que entra em “Selecionados”) até o momento em que o item é puxado pela próxima coluna. Quanto maior o seu lote de reabastecimento (limite da coluna “Selecionados”), mais alto é o tempo de reação. É interessante acompanhar essa métrica para entender se os itens estão ficando muito tempo “parados” na fila de entrada. A diferença entre o tempo de reação e o lead time é tempo de ciclo. Veja abaixo:



Além disso, podemos medir o tempo de ciclo de um conjunto de colunas específicas, como indicado na figura (cycle time dev-testes).

Composição do fluxo

Como vimos nos elementos de um sistema Kanban, você pode designar diferentes “tipos” para os itens do seu quadro. O mais comum é analisar a quantidade de “novos itens de valor” (funcionalidades) vs a quantidade de “itens de defeito”. Esses diferentes perfis determinam a composição do fluxo, que você pode acompanhar ao longo do tempo:



### 4.4 - SCRUM

1. Visão Geral da Metodologia Ágil Scrum

O Scrum não é um processo padronizado onde metodicamente você segue uma série de etapas sequenciais e que vão garantir que você produza, no prazo e no orçamento, um produto de alta qualidade e que [encanta os seus clientes](http://www.mindmaster.com.br/como-vender-scrum/).

Em vez disso, o Scrum é um framework para organizar e gerenciar trabalhos complexos, tal como projetos de desenvolvimento de software.

O framework Scrum é um conjunto de valores, princípios e práticas que fornecem a base para que a sua organização adicione suas práticas particulares de engenharia e gestão e que sejam relevantes para a realidade da sua empresa. O resultado será uma versão de Scrum que é exclusivamente sua.

Para melhor entender este conceito, imagine que o framework seja como a fundação e as paredes de um edifício. Os valores do Scrum, princípios e práticas seriam os principais componentes estruturais. Você não pode ignorar ou mudar fundamentalmente um valor, princípio ou prática sem o risco de colapso.

O que você pode fazer, porém, é personalizar o interior da estrutura do Scrum, acrescentando artefatos e recursos até que você tenha e um processo que [funciona para sua empresa](http://www.mindmaster.com.br/como-implantar-scrum/).

1. Valores

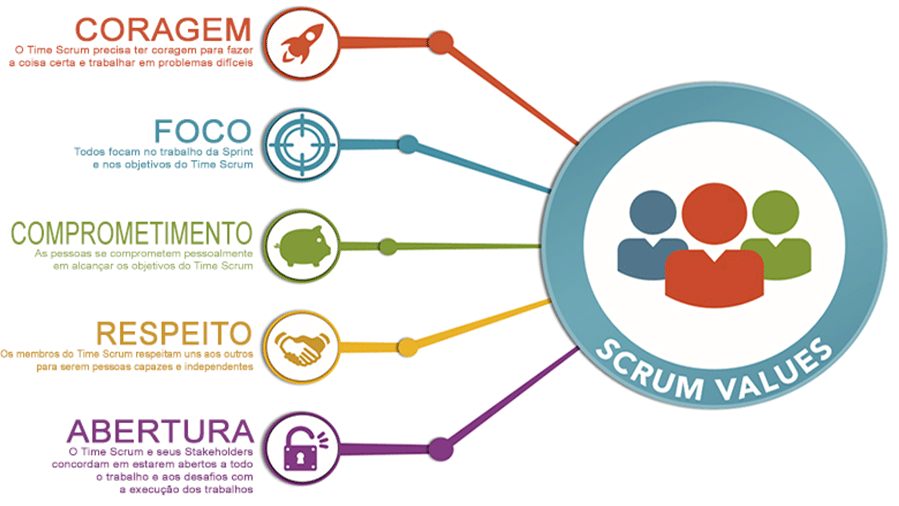


Figura 8: Valores Scrum

1. A Base Fundamental

A base fundamento é composta pelas seguintes práticas:

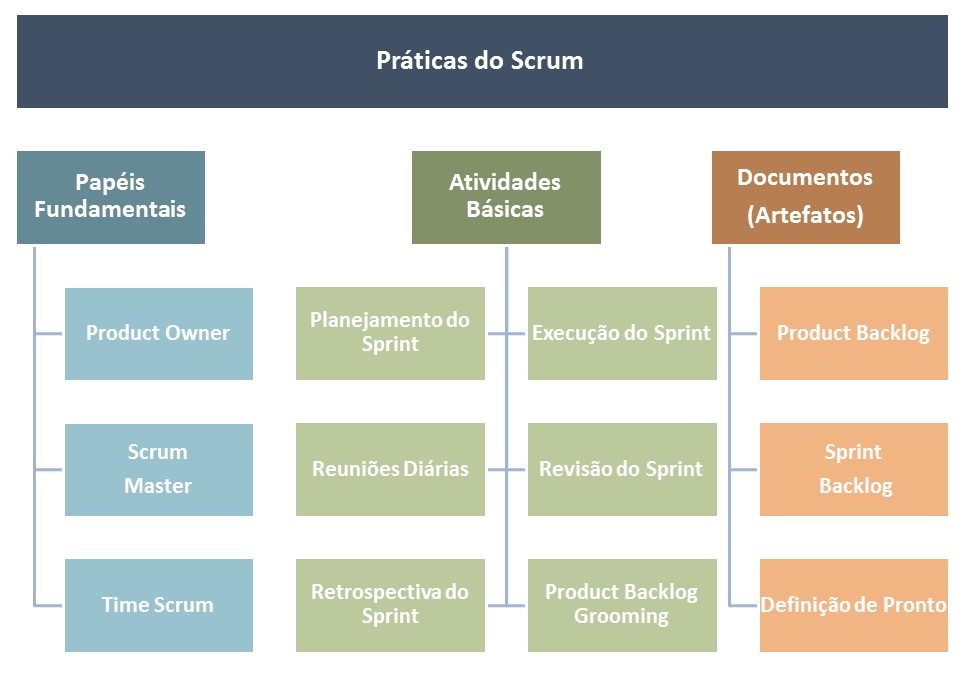


Figura 9: Bases Scrum

1. Papéis Fundamentais

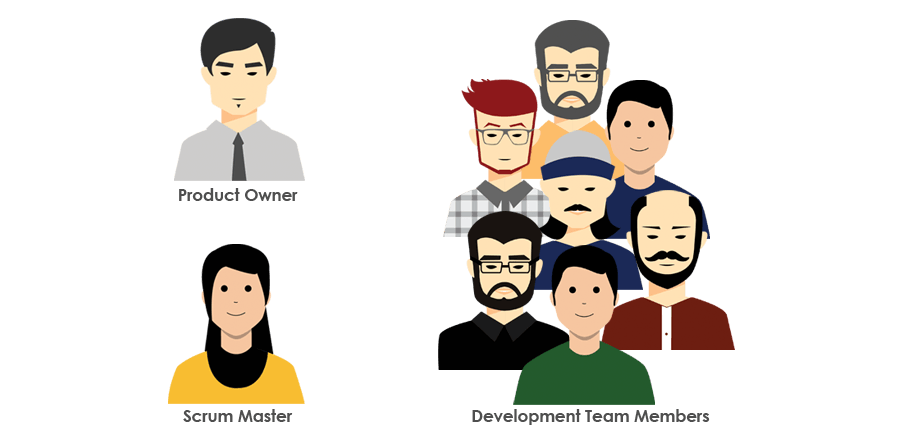


Figura 10: Equipe Scrum

Os esforços de desenvolvimento utilizando Scrum, consiste em uma ou mais equipes Scrum, cada uma composta basicamente de três papéis:

* [Product Owner](http://www.mindmaster.com.br/product-owner/);
* [ScrumMaster](http://www.mindmaster.com.br/scrum-master);
* [Time de Desenvolvimento](http://www.mindmaster.com.br/time-scrum/).

Podem haver outros papéis ao usar Scrum, mas o framework básico requer apenas os três listados aqui.

* [Product Owner](http://www.mindmaster.com.br/product-owner/): é o ponto central com poderes de liderança sobre o produto. Ele é o único responsável por decidir quais recursos e funcionalidades serão construídos e qual a ordem que devem ser feitos.

É responsabilidade dele manter e comunicar a todos os outros participantes uma visão clara do que a equipe Scrum está buscando alcançar no projeto. Como tal, ele é responsável pelo sucesso global da solução.

Para garantir que a equipe construa rapidamente o que o [Product Owner](http://www.mindmaster.com.br/product-owner/) precisa, ele deve colaborar ativamente com o [Scrum Master](http://www.mindmaster.com.br/scrum-master) e [equipe de desenvolvimento](http://www.mindmaster.com.br/time-scrum/) e deve estar disponível para responder às perguntas tão logo estas são feitas.

* O [Scrum Master](http://www.mindmaster.com.br/scrum-master) é responsável por ajudar a todos os envolvidos a entender e abraçar os valores, princípios e práticas do Scrum.

Ela age como um *Coach*, executando a liderança do processo e ajudando a equipe Scrum (e o resto da organização) a desenvolver sua própria abordagem do Scrum, que tenha a melhor performance, respeitando as particularidades da organização.

O [ScrumMaster](http://www.mindmaster.com.br/scrum-master) também tem um papel de facilitador.  Ele deve ajudar a equipe a resolver problemas e fazer melhorias no uso do Scrum. Ele também é responsável por proteger a equipe contra interferências externas e assume um papel de liderança na remoção de impedimentos que podem atrapalhar a produtividade.

Normalmente o ScrumMaster não tem autoridade para exercer o controle sobre a equipe, de modo que este papel não é o  
mesmo que o papel tradicional do Gerente de Projeto ou Gerente de Desenvolvimento. O ScrumMaster age como um líder, não como um gerente.

* No desenvolvimento tradicional de software são abordados vários tipos de trabalho, tais como: arquiteto, programador, testador, administrador de banco de dados, Designer, e assim por diante.

No Scrum é definido o papel do [Time de Desenvolvimento](http://www.mindmaster.com.br/time-scrum/), que é simplesmente a junção de todas essas pessoas em uma equipe multidisciplinar, e que são responsáveis pela concepção, construção e testes do produto.

A ideia principal é que a equipe de desenvolvimento se auto organiza para determinar a melhor maneira de realizar o trabalho para atingir a meta estabelecida pelo Product Owner.

Um time de desenvolvimento tem tipicamente entre **5 e 9 pessoas** e seus membros devem ter coletivamente todas as habilidades necessárias para produzir, com qualidade, software funcionando.

Claro, scrum pode também ser usado em projetos que exigem equipes muito maiores. No entanto, ao invés de ter uma equipe Scrum com, digamos, 30 pessoas, seria melhor ter entre 3 ou mais times scrum, cada um com um time de 9 ou menos pessoas.

Abaixo outro tipo de imagem que é também muito utilizada para representar as interações entre as atividades no processo.

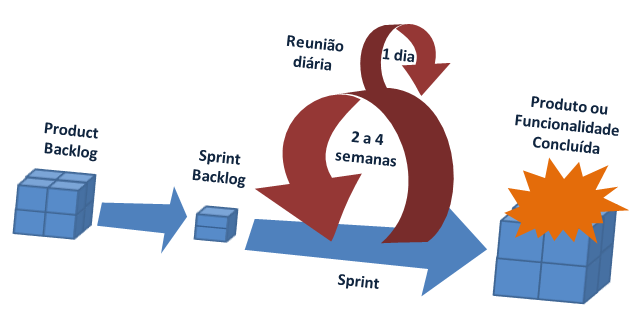


Figura 11: Processo Scrum

O [Product Owner](http://www.mindmaster.com.br/product-owner/) tem uma visão do que ele quer criar (o grande cubo). Como o o cubo pode ser grande, por meio de uma atividade chamada Grooming, ele é dividido em um conjunto de funcionalidades que são compilados em uma única lista priorizada chamado de **Product Backlog**.

Então é feito a primeira reunião de **Planejamento de Sprint**, para definir o **Sprint Backlog,**que contém todo o trabalho que será executado durante o **Sprint.**

O **Sprint** tem duração média de **2 a 4 semanas** e são feitas reuniões diárias de acompanhamento (Daily Scrum) do trabalho.

1. **Product Backlog**

No Scrum, sempre fazemos o trabalho mais importante primeiro.

O [Product Owner](http://www.mindmaster.com.br/product-owner/), com ajuda do resto da equipe Scrum e as partes interessadas, é o responsável por determinar e gerir a sequência deste trabalho e comunicando-o na forma de uma lista de prioridades conhecida como o **Product Backlog.**

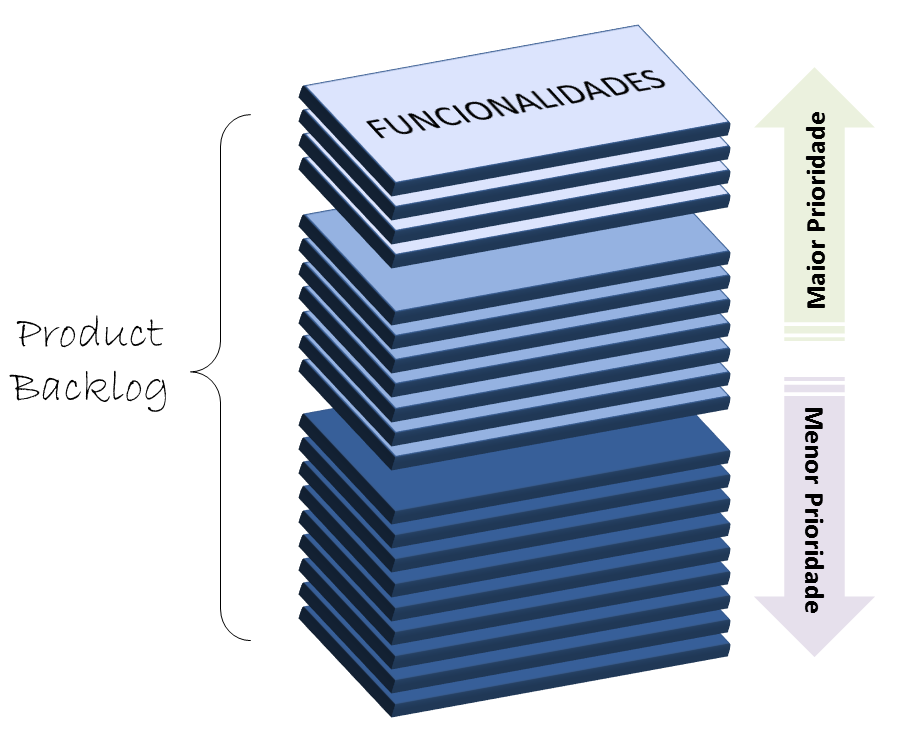


Figura 12: **Product Backlog**

 O Product Owner, em conjunto com as demais partes interessadas no produto, definem os itens do Product Backlog.

Em seguida, ele garante que os itens do Backlog sejam colocados na sequência correta (usando fatores como valor, custo, conhecimento e risco), de modo que os itens de alto valor apontem no topo do backlog do produto e os itens de menor valor em direção ao fundo.

O Product backlog é um documento que está constantemente evoluindo. Os itens podem ser adicionados, excluídos e revisto pelo Product Owner por conta de mudanças nas condições de negócios, ou conforme a compreensão da equipe Scrum sobre o produto aumenta.

Em geral a atividade de criar e de refinar os itens do product backlog, estimando o tamanho e esforço de cada item, é chamada de Grooming.

Antes de finalizar a priorização, ou refinamento do produto backlog, é preciso saber o tamanho de cada item. É importante que o Product Owner saiba o custo de cada item para que possa determinar a sua prioridade de forma adequada. O Scrum não especifica como você deve medir o tamanho dos itens do backlog.

Na prática, muitas equipes usam uma medida de tamanho relativo, como Story Point ou dias ideais.

1. **Sprints**

No Scrum, o trabalho é realizado em iterações ou ciclos de até um mês de calendário chamado de **Sprints**.

O trabalho realizado em cada sprint deve criar algo de valor tangível para o cliente ou usuário. Sprints são **timeboxed** (duração fixa) para que tenham sempre um início e fim data fixa, e, geralmente, todos eles devem estar com a mesma duração.

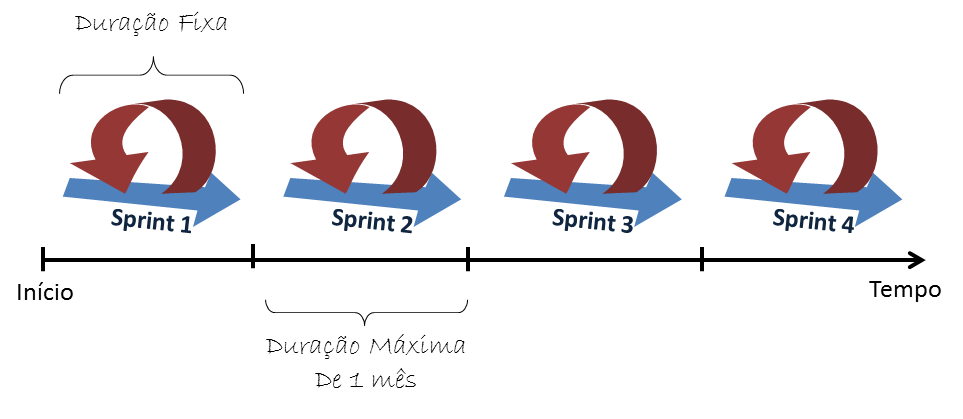


Figura 13: **Sprints**

Um novo Sprint segue imediatamente a conclusão do Sprint anterior e, via de regra, não devemos permitir nenhuma alteração de escopo ou pessoal durante um Sprint (mas por experiência própria posso afirmar que esta regra é quase sempre quebrada devido à algumas necessidades de negócio)

1. **Sprint Planning**

O product backlog pode representar muitas semanas ou até meses de trabalho, o que é muito mais do que pode ser concluído em um único e curto sprint.

Para determinar quais os subconjuntos de itens do Product Backlog mais importantes para construir no próximo sprint, o product owner, junto com o [time de desenvolvimento](http://www.mindmaster.com.br/time-scrum/) e [Scrum Master](http://www.mindmaster.com.br/scrum-master), devem realizar o Sprint Planning (planejamento de sprint ).

Durante o planejamento do sprint, a equipe de desenvolvimento e o product owner devem chegar a um acordo sobre qual o Objetivo do Sprint.

Com este objetivo em mãos, eles determinam quais os itens do backlog devem ser priorizados para serem executados neste Sprint.

A maioria das equipes Scrum que estão realizando Sprints de duas semanas a um mês de duração tentam completar o planejamento do sprint em cerca de 4 a 8 horas.

Um sprint de uma semana não deve tomar mais do que 2 horas para planejar.

1. **Daily Scrum**

Todos os dias, idealmente no mesmo horário, os membros da equipe de desenvolvimento devem realizar uma reunião com tempo definido (15 minutos ou menos), chamado Daily Scrum.

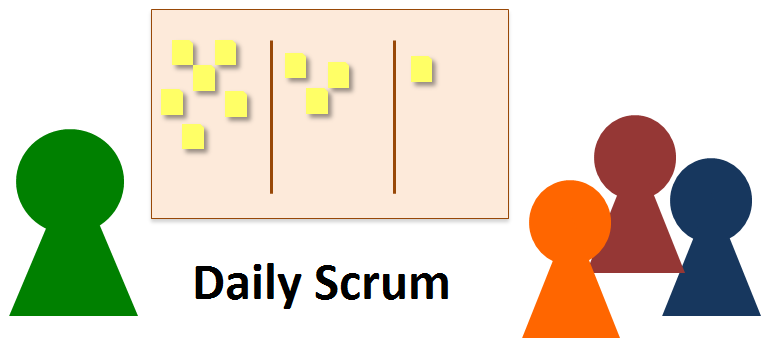


Figura 14: **Daily Scrum**

Esta reunião também é muitas vezes chamada de Stand-Up Meeting, por causa de uma prática recomendada para que a reunião seja feita em pé(com a intenção de fazer com que a reunião seja rápida).

Uma abordagem comum nesta reunião é o [Scrum Master](http://www.mindmaster.com.br/scrum-master) perguntar para cada membro da equipe três perguntas:

3 Perguntas básicas da Reunião Diária:

* O que fiz ontem que ajudou o time a atingir a meta do sprint?
* O que vou fazer hoje para ajudar o time a atingir a meta do sprint?
* Existe algum impedimento que não permita a mim ou ao time atingir a meta do sprint?

Ao responder a estas questões, todos conseguem visualizar de uma maneira geral como está progredindo o trabalho do Sprint em direção à meta.

1. **Definition of Done (Definição de Pronto)**

No Scrum nós consideramos como resultado do Sprint produto ou funcionalidade concluída.

Para saber quando, e como, uma parte do produto ou funcionalidade deve ser considerada concluída nós utilizamos um documento chamado Definition of Done.

Para aprender mais sobre DoD, [clique aqui](http://www.mindmaster.com.br/definition-of-done/) e leia o nosso artigo completo sobre o assunto.

Embora, isso varie significativamente de um extremo ao outro para cada time Scrum, os integrantes devem ter um entendimento compartilhado do que significa o trabalho estar completo, assegurando a transparência. Esta é a **“**Definição de Pronto**”** para o Time Scrum e é usado para assegurar quando o trabalho esta completado no incremento do produto.

1. **Sprint Review (Revisão do Sprint)**

No final do Sprint, existem duas atividades adicionais que são fundamentais. Uma delas é chamada Sprint Review.

O objetivo desta atividade é verificar e adaptar o produto que está sendo construído.

Esta é uma reunião informal, e a apresentação do incremento destina-se a motivar e obter comentários e promover a colaboração.

1. **Sprint Retrospective (Retrospectiva do Sprint)**

Enquanto o objetivo do Sprint Review é verificar necessidades de adaptações no produto, o Sprint Retrospective tem como objetivo verificar necessidades de adaptações no processo de trabalho.

A Retrospectiva do Sprint ocorre depois da Revisão da Sprint e antes da reunião de planejamento da próxima Sprint. Esta é uma reunião time-boxed de três horas para uma Sprint de um mês.

1. Benefícios do scrum:

* Entregas frequentes de valor

Ao contrário do método tradicional de desenvolvimento de software, com o Scrum nós devemos focar em realizar entregas de alto valor ao cliente. Ou seja, priorizamos as funcionalidades de maior valor de negócio para que, antes mesmo do desenvolvimento do produto terminar, já possamos entregar algo palpável e utilizável para os clientes. Isso permite o lançamento de um produto que atende as necessidades mais valiosas dos usuários em menor tempo.

* Redução dos riscos do projeto

O Scrum possibilita a redução dos riscos de negócios do projeto graças a, entre outras coisas, a colaboração entre todos os interessados no projeto, o desenvolvimento dividido em ciclos curtos e as entregas frequentes de valor através de funcionalidades importantes sendo desenvolvidas e entregues primeiro. Isso garante um feedback mais assertivo dos usuários acerca do que está sendo desenvolvido a cada ciclo. Caso as expectativas do cliente não estejam sendo satisfeitas, o problema é rapidamente identificado e podemos corrigir o rumo do projeto.

* Mudanças utilizadas como vantagem competitiva

Um problema no negócio do projeto pode ser rapidamente identificado e corrigido porque, ao contrário dos métodos tradicionais, o Scrum é aberto à mudanças. No Scrum, os planos e as especificações de negócio não são totalmente definidas no início do projeto, elas são trabalhadas e aprimoradas em todo o decorrer do desenvolvimento, isso permite que mudanças sejam feitas em qualquer estágio. Desta forma, podemos garantir que o produto entregue realmente tem valor para os clientes e está acompanhando as mudanças do mercado.

* Maior qualidade do produto

O time Scrum é responsável em entregar seus produtos com a maior qualidade possível. Como o desenvolvimento e as entregas são incrementais, podemos validar continuamente o produto entregue e identificar erros e outros problemas antes que se tornem problemas maiores e de difícil tratamento, podendo causar atrasos no projeto.

* Transparência no progresso do projeto

Através de diversas práticas do Scrum, todos os envolvidos no desenvolvimento do produto possuem total visibilidade de seu progresso. Por exemplo, através do Daily Scrum – uma reunião diária de poucos minutos – os integrantes do time respondem a três perguntas fundamentais que farão com que todo o time saiba o que se passa no projeto e quais os próximos passos até a próxima reunião: O que eu fiz desde a última reunião de Daily Scrum? O que pretendo fazer até a próxima reunião? O que está impedindo a realização de meu trabalho?

Outra prática que traz maior transparência ao desenvolvimento do produto é o quadro Scrum. Neste quadro é possível visualizar o andamento do ciclo atual, já que todas as tarefas estão dispostas em uma das três fases do quadro: To Do (À Fazer), Doing (Fazendo) e Done (Feito).

* Redução do desperdício

O Scrum nos orienta a buscar a simplicidade utilizando e produzindo apenas o que é necessário e suficiente para desenvolver nossos produtos.

* Aumento da produtividade

A produtividade é potencializada nos times que utilizam Scrum por diversos fatores, incluindo: o trabalho em equipe e a autonomia do time; a remoção de impedimentos; a melhoria contínua dos processos; e a maior motivação do time.

## 5 – Referências Bibliográficas

http://www.mindmaster.com.br/scrum/

https://www.devmedia.com.br/introducao-ao-fdd-feature-driven-development/27971

http://www.digisystem.com.br/7-beneficios-do-scrum-no-desenvolvimento-dos-produtos-digisystem/

https://targetteal.com/pt/blog/metodo-kanban/

<https://www.culturaagil.com.br/kanban-do-inicio-ao-fim/>

https://www.ft.unicamp.br/liag/Gerenciamento/monografias/monogafia\_metodos\_ageis.pdf

<http://www.fatecsp.br/dti/tcc/tcc00064.pdf>

https://www.projectbuilder.com.br/blog/quais-sao-os-principais-tipos-de-metodos-ageis/

https://www.devmedia.com.br/processo-de-teste-agil-x-tradicional/36854

https://www.agilealliance.org/glossary/xp/#q=~(infinite~false~filters~(postType~(~'post~'aa\_book~'aa\_event\_session~'aa\_experience\_report~'aa\_glossary~'aa\_research\_paper~'aa\_video)~tags~(~'xp))~searchTerm~'~sort~false~sortDirection~'asc~page~1)

https://blog.mastertech.tech/negocios/extreme-programming-metodologia-agil-para-o-desenvolvimento-de-software/

https://www.devmedia.com.br/introducao-ao-fdd-feature-driven-development/27971

http://www.fabiocruz.com.br/valores-scrum/

http://tdd.caelum.com.br/

http://www.agileandart.com/2010/07/16/ddd-introducao-a-domain-driven-design/

https://blog.egestor.com.br/o-que-e-e-como-funciona-o-metodo-kanban/

https://sistemasoma.com.br/blog/empreendedorismo/a-historia-do-kanban-e-como-ele-pode-ajudar-a-aumentar-a-produtividade