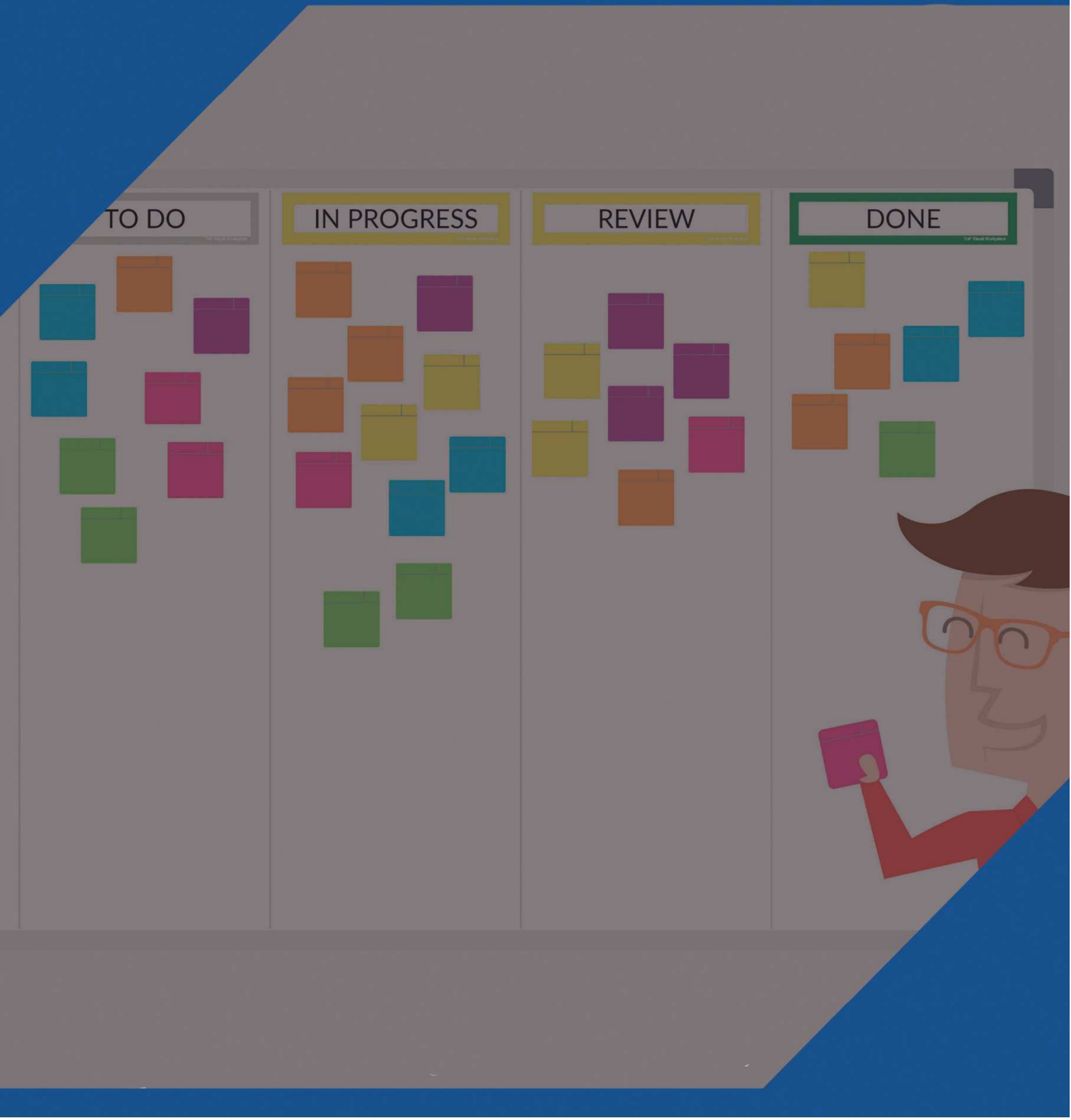
DISCIPLINA DE
METODOLOGIAS ÁGEIS



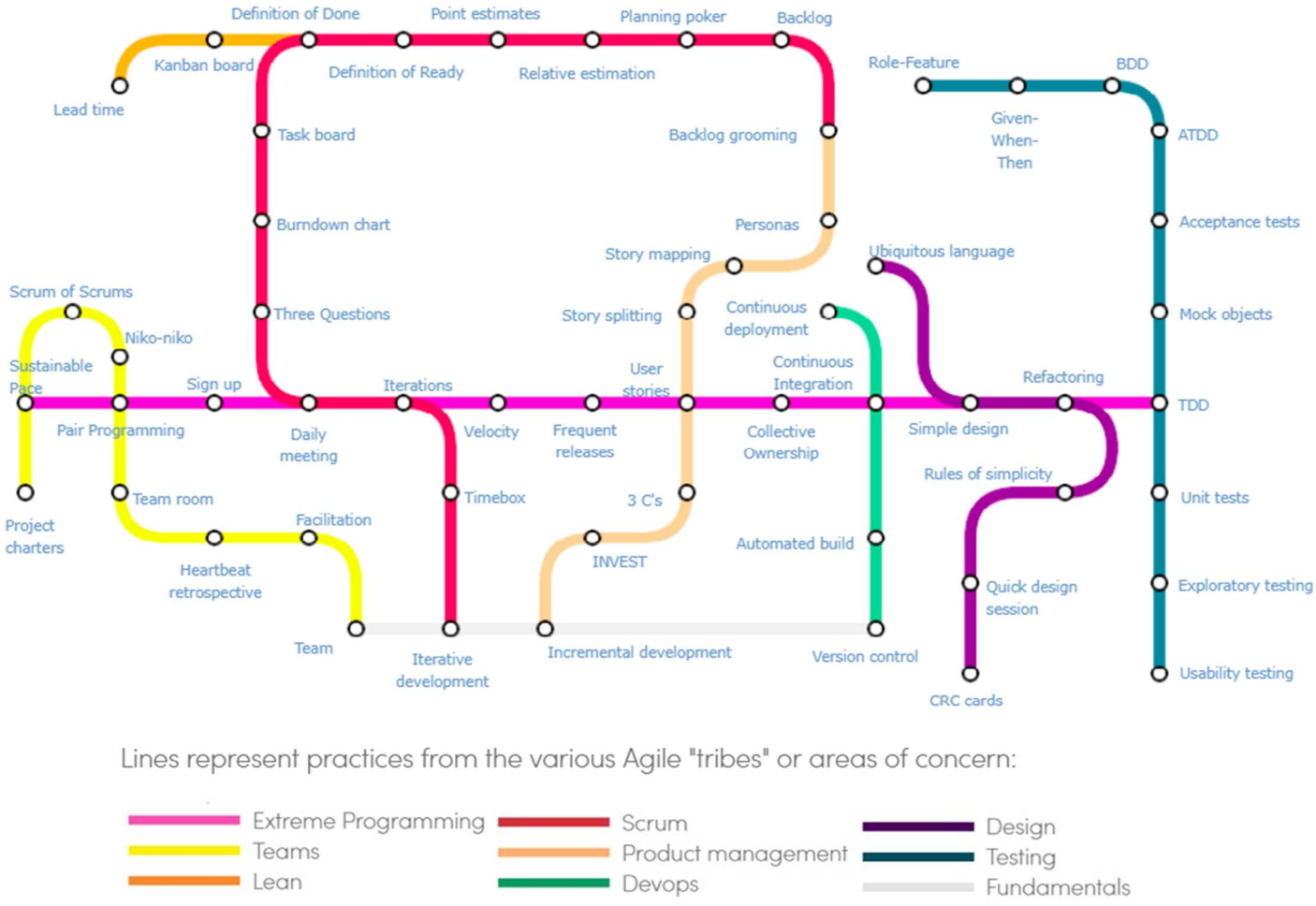


Figura 1 - O mapa da agilidade

Sumário

Apresentação

Prof. Esp. Wagner Mendes Voltz

Olá, este é o livro de **metodologias ágeis**, que foi elaborado especialmente para você conhecer ou aprimorar as diversas práticas e ferramentas no desenvolvimento de software. O conteúdo deste livro se aplica a codificadores, gerentes de projetos, implantadores, analistas, testadores e donos de produtos (que já é uma terminologia ágil).

Meu nome é Wagner Mendes Voltz e sou o autor deste livro. Minha formação é em Tecnologia em Informática pela Universidade Federal do Paraná (UFPR) e o ano de conclusão desta graduação foi em 2005.

Antes mesmo de estar formado, já participava de programas de estágio na área de desenvolvimento de sistemas, utilizando a linguagem de programação PHP.

Em 2006, já atuando como analista de sistema, percebi a dificuldade que eu tinha em gerenciar equipes de desenvolvimento e para aprender mais sobre gestão de pessoas, ingressei numa especialização oferecida por uma escola de negocias (FAE Business School). Concluí este curso em 2007, com o título de Especialista em Administração e Gestão da Informação.

Creio que esta especialização trouxe uma nova visão de como trabalhar com pessoas e computadores e o desejo de lecionar já era existente em 2007.

Em 2008, comecei a desenvolver software na linguagem de programação Delphi, utilizando banco de dados Oracle e em 2010, dediquei todos os meus esforços para investir no aprendizado e desenvolvimento de aplicativos em Java, que atualmente é a minha especialidade.

Com o foco na linguagem Java e em desenvolvimento de sistemas, pude participar de diversos projetos e desafios, os quais me levaram a aprofundar o conhecimento de orientação a objetos e testes unitários. Mas alguma coisa ainda não ia bem, pois estávamos fazendo software, mas de forma descontrolada e desorganizada. Mesmo usando modelos tradicionais da engenharia de software, parecia que não saíamos do lugar.

Em 2011, tive o meu primeiro contato com agilidade. Uma consultoria, que visitou o lugar onde eu trabalhava, nos apresentou o Scrum. Gostei tanto que investi tempo esforço para me certificar neste *framework*, com isto me tornei um CSM (*Certified Scrum Master*) pela Scrum Alliance.

Com o passar dos anos fui percebendo a necessidade de aprender outros “sabores” da agilidade e fui sendo apresentado para a imagem que está no início do livro.

Aprendi um pouco de XP (*eXtreme Programming*), Kanban, métricas, *clean code* (código limpo), entre outros. Nesta caminhada, tive oportunidades de palestrar no Agile Tour de Maringá-PR, Agile Brasil 2016 e no TDC (*The Developers Conference*). Além disto, fiz vários amigos que estão fazendo uma grande mudança na produção de software nacional, gerando valor para o cliente.

Hoje estudo de tudo um pouco e amplio a minha caixa de ferramentas e tento identificar qual é o momento para usar cada um dos aprendizados.

Espero que este livro possa lhe ajudar a conhecer um pouco mais deste assunto.

Caso queira me acompanhar, me adicione em alguma rede social para trocarmos figurinhas:

- Twitter: @tiofusca
- Blog: http://medium.com/@wagnerfusca/
- Linkedin: https://www.linkedin.com/in/wagnervoltz/

E aí, está preparado? Vamos nessa??!!

Capítulo 1 - Mergulhando nos princípios ágeis

Como surgiu

A tecnologia é parte do nosso dia a dia e para interagirmos com ela, faz-se necessário o uso de diversos dispositivos. Como exemplo, podemos citar os notebooks, *smartphones* e *tablets*. E para cada um destes dispositivos temos pelo menos um software em funcionamento. E como cada um de nós tem uma necessidade, existe uma imensa oportunidade na produção de softwares para atender esta demanda. Talvez esta seja a indústria que mais cresceu nos últimos anos.

A produção de um software não é tão simples quanto parecer ser. E nem tão pouco é um trabalho repetitivo, como o de um operário em uma indústria. A produção de software está muito mais próxima da criação de uma obra criativa (como um quadro ou música), do que o trabalho fabril executado em fábricas.

Visando esta complexidade, surgiram vários modelos para o desenvolvimento de software, dentre eles:

- Modelo cascata (*waterfall*)
- Modelo espiral
- Modelo iterativo e incremental
- Processo unificado (RUP)

Os modelos acima citados têm suas características e premissas, mas muitas das vezes eles não atendem a velocidade que a indústria de software requisita. Por exemplo, o modelo em cascata prevê que todos os requisitos estejam definidos e escritos para daí começar a etapa seguinte, a de elaboração do projeto. Em sequência começa a implementação e codificação dos requisitos. Após esta etapa, começam os testes e em sequência a implantação do software. Com isto, chegamos ao fim do processo. Uma nova rodada destes passos deve começar, repetindo todas as etapas.

A figura 2 está representando este ciclo. Ele funciona e atende as necessidades da criação de um software, mas ele tende a ser muito lento, dependendo da quantidade de requisitos definidos no primeiro passo. Com isto, começamos a ter entrega de software mais demorada e com muitas alterações. Nada garante que a alteração solicitada ainda faça sentido, pois muito tempo pode ter se passado da data de solicitação e atendimento da demanda.

Estes modelos tradicionais não trabalham com a hipótese de mudança e se te uma coisa que muda são os requisitos e as necessidades dos usuários do software. Por exemplo, no modelo em cascata, uma mudança, só pode ser realizada, quando o primeiro ciclo de implementação tenha sido finalizado. A mudança irá gerar novos requisitos, nova mudança no projeto, nova codificação, novos testes e nova versão. Esta sequência de passos tende a ser lenta e não desejada neste modelo, pois ela causa mudanças severas.

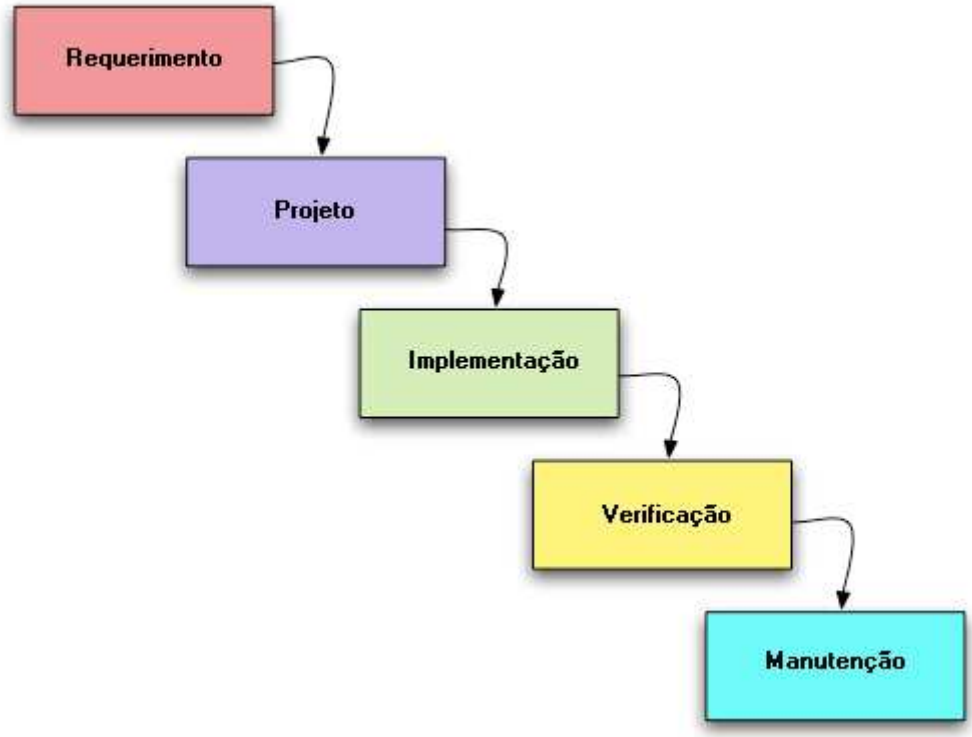


Figura 2- Modelo em cascata

Para não ficarmos somente na teoria, vamos olhar alguns fatos identificados em projetos que estavam usando estes modelos tradicionais. Para isto, vamos considerar um estudo que o Standish Group, uma empresa localizada em Massachusetts, EUA, publica desde 1994 e permanece até os dias atuais. Este estudo é chamado *CHAOS Report*.

O CHAOS Report do ano 2000 (THE STANDISH GROUP INTERNATIONAL, 2001) apresenta uma coletânea de dados envolvendo 280 mil projetos de software nos Estados Unidos, os quais revelaram que:

- Em média, os atrasos representaram 63% mais tempo do que o estimado;
- Os projetos que não cumpriram o orçamento custaram em média 45% mais e
- No geral, apenas 67% das funcionalidades prometidas foram efetivamente entregues.

Ao analisarmos estes dados, foi definido um cenário denominado **Crise do Software**.

O Standish Group classifica o resultado final de um projeto nas seguintes categorias:

- Mal sucedido;
- Comprometido e
- Bem sucedido.

Em 2000, o resultado final da pesquisa mostrou a seguinte distribuição entre os projetos:

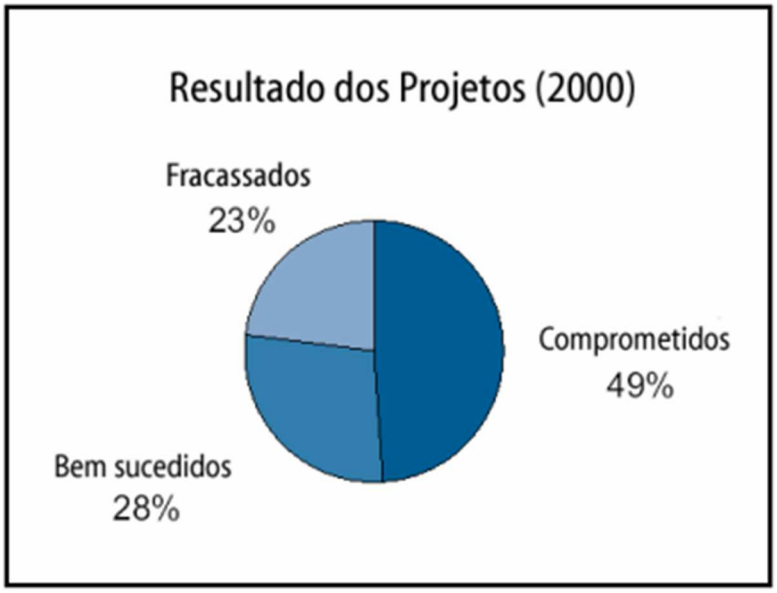


Figura 3 - Resultados dos projetos de software em 2000

E num comparativo de 1994 até 2000, percebemos uma evolução no que se tem se chamado **crise do software**.

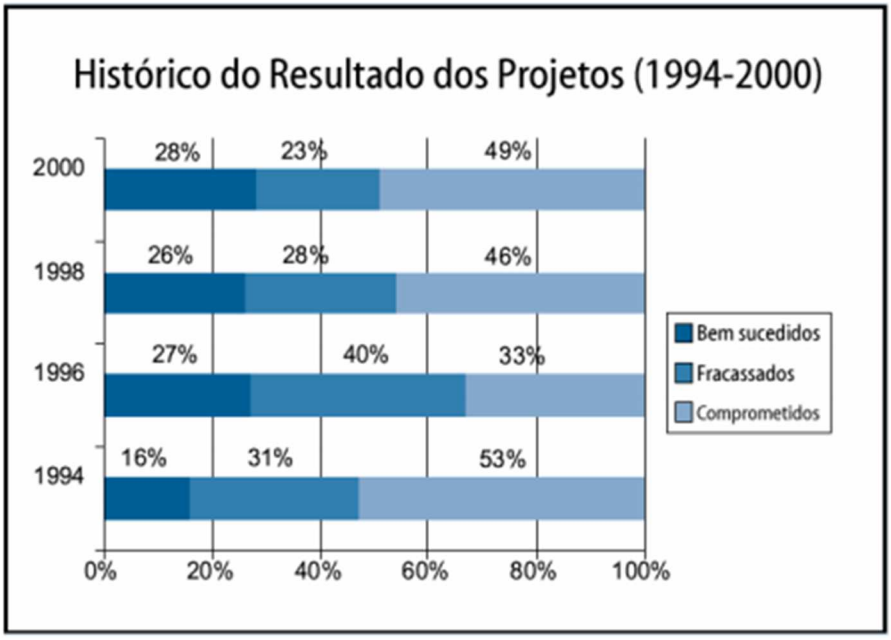


Figura 4 - Histórico dos indicadores dos projetos de software entre 1994 e 2000

Mas outro dado um tanto quanto revelador, foi apresentado na terceira Conferência Internacional sobre *Extreme Programming*, que ocorreu na Itália em 2002. Jim Johnson, presidente do Standish Group, apresentou um estudo revelador sobre a utilização das funcionalidades nos projetos que foram pesquisados pelo Standish Group (JOHNSON, 2002). Os resultados demonstram que 45 por cento das

funcionalidades encontradas em um sistema típico jamais são usadas e 19 por cento raramente são usadas:

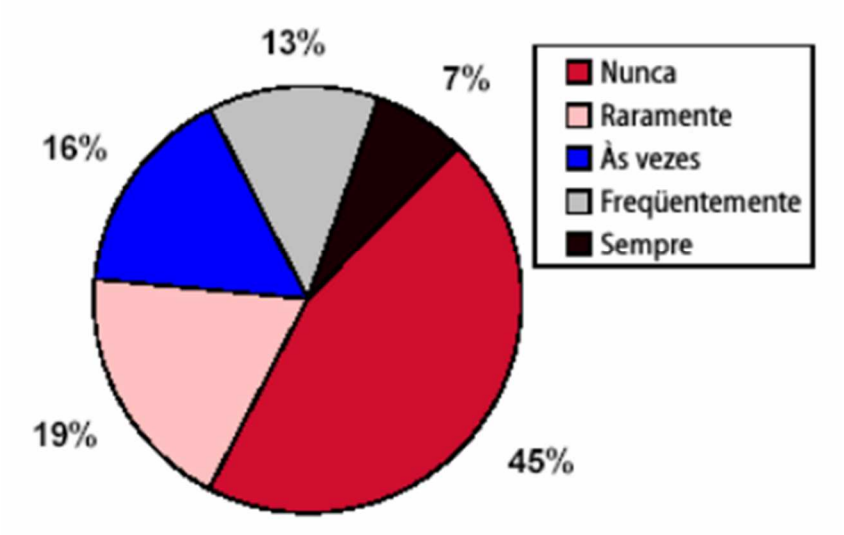


Figura 5 - Indicador de uso de funcionalidades num sistema

Os dados são de mais de 10 anos atrás, mas o relatório CHAOS continua sendo publicado e os comportamentos de muitos projetos ainda estão semelhantes aos reportados há 10 anos. Os projetos que tem melhorado o desempenho deve-se, em especial, a adoção de metodologias ágeis. Para entender um pouco mais da diferença do modelo tradicional para o ágil, Prikladnicki, Willi e Milani (2014) elaborou o seguinte quadro:

	TRADICIONAL	METODOLOGIAS ÁGEIS
Pressupostos fundamentais	Sistemas totalmente especificáveis, previsíveis; desenvolvidos a partir de um planejamento extensivo e meticuloso	Software adaptativo e de alta qualidade; pode ser desenvolvido por equipes pequenas utilizando os princípios da melhoria contínua do projeto e testes orientados a rápida resposta a mudanças
Controle	Orientado a processos	Orientado a pessoas
Estilo de gerenciamento	Comandar e controlar	Liderar e colaborar
Gestão do conhecimento	Explícito	Tácito
Atribuição de papéis	Individual – favorece a especialização	Times auto-organizáveis – favorece a troca de papéis
Comunicação	Formal	Informal
Ciclo do projeto	Guiado por tarefas ou atividades	Guiado por funcionalidades do produto
Modelo de desenvolvimento	Modelo de ciclo de vida (Cascata, Espiral, ou alguma variação)	Modelo iterativo e incremental de entregas
Forma/estrutura organizacional desejada	Mecânica (burocrática com muita formalização)	Orgânica (flexível e com incentivos a participação e cooperação social)

Figura 6 - Diferença modelo tradicional e metodologias ágeis

O que as metodologias ágeis não são e não pregam

1) Metodologia ágil não é o *"lado negro da força"*

   Para alguns, agilidade é um grupo de rebeldes que querem um espaço para ser ouvido.

2) Metodologia ágil não é PMBOK

   PMBOK é um guia muito completo para gerenciamento de projetos. Ele não deve ser descartado, mas devem ser analisados criteriosamente, quais itens fazem sentido para o projeto.

3) Metodologia ágil não é garantia de fazer mais rápido

   Agilidade é confundida com entregar mais ou entregar mais rápido. **Não!!!** Este pensamento está errado. Agilidade é ser mais assertivo e realmente fazer o que precisa ser feito.

4) Metodologia ágil não é o fim da documentação.
   Deve existir documentação num projeto ágil, mas só o que faz sentido.
5) Metodologia ágil não serve para processos

   Este pensamento é errado. Nos próximos capítulos você vai identificar que processos estão presentes nas metodologias ágeis, mas de maneira eficaz e com ciclos de *feedback* mais curtos.

6) Metodologia ágil não tem métrica

Nada pode ser melhorado se não tiver sido medido. A inspeção é um item muito presente nas metodologias ágeis.

7) Metodologia ágil é somente para os desenvolvedores

Não! Não é só para desenvolvedores.

8) Metodologia ágil é só para empresa pequena

Não! Gigantes da tecnologia, como a IBM, já entenderam isto e adotaram as metodologias ágeis em suas empresas.

O que as metodologias ágeis são e pregam

A metodologia ágil consiste em ciclos curtos e *feedback* constante do cliente. Além disso, promove a integração e colaboração de todos os envolvidos. O gerenciamento é no fluxo, não na atividade. E o foco está na entrega de valor.

Metodologia ágil prega:

- Evitar o desperdício de fazer mais do que o necessário
- Entrega de valor ao cliente (entregar o que vai ser usado)

Para entender mais o que é pregado, leia o manifesto ágil, que é foi elaborado por 17 engenheiros de software e neste manifesto encontramos os princípios que fundamentam o desenvolvimento ágil de software.

Manifesto Ágil

Foi publicado em 2001 por um grupo de agilistas. Seu conteúdo está disponível em www.manifestoagil.com.br e pode ser conferido abaixo:

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

1. ***Indivíduos e interações*** *mais que processos e ferramentas*
2. ***Software em funcionamento*** *mais que documentação abrangente*
3. ***Colaboração com o cliente*** *mais que negociação de contratos*
4. ***Responder a mudanças*** *mais que seguir um plano*

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

Os doze princípios

No mesmo site onde encontramos o Manifesto Ágil, é possível identificar os princípios que embasam o manifesto. São eles:

- *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 à 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, aumenta a agilidade.*

- *Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.*

- *As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis.*

- *Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.*

Fonte: http://www.manifestoagil.com.br/principios.html

Agora que você entendeu o cenário denominado **crise de software**, e os princípios das metodologias ágeis, vamos começar a se aprofundar no tema. Começaremos pelo SCRUM que é conjunto de práticas bem definidas e que podem mudar todo o seu ambiente de desenvolvimento de software, chegando assim próximo de um modelo ágil.

Capítulo 2 – Scrum

Figura 7 – a posição scrum no rugby

Princípios e fundamentos

O termo Scrum é muito comum para um jogador de rugby. Ele representa uma formação que ambos os times, unidos, vão em busca da bola. A formação Scrum deve ser feita para reiniciar uma partida que foi para por uma jogada irregular ou penalização.

A figura acima mostra a posição Scrum sendo realizado poucos momentos antes da bola ser colocada em disputa.

Em 1986, Takeuchi e Nonaka publicaram um estudo na Harvard Business Preview que comparavam equipes de alto desempenho e multidisciplinares com a formação scrum do rugby. A conclusão deste estudo é que equipes pequenas e multidisciplinares produziam os melhores resultados.

Em 1993, houve a primeira concepção, documentação e implementação do Scrum numa empresa e com isto surgia este framework ágil. Os seus primeiros implementadores foram Jeff Sutherland, John Scumniotales e Jeff McKenna. Em 1995, numa parceria entre Jeff Sutherland e Ken Schwaber, houve a formalização do processo para a indústria mundial de software. Esta formalização foi apresentada na conferência OOPSLA (*Conference on Object-Oriented Programming Systems, Languages, and Applications*).

A partir daí, o Scrum tornou-se a metodologia mais empregada no mundo ágil. Como características do Scrum temos conceitos de Lean, desenvolvimento iterativo e incremental, teoria das restrições, teoria de sistemas adaptativos complexos e do

estudo de Takeuchi e Nonaka, somadas com experiências positivas vividas por profissionais de desenvolvimento de software.

O Scrum objetiva a maximização de entrega de software de modo eficaz, adaptando-se a realidade das mudanças.

O Scrum é denominado um framework e não uma metodologia. Uma metodologia é o estudo de métodos e técnicas embasados cientificamente e que são utilizadas em processos de investigação. Diferente de um processo mais tradicional, o Scrum não irá te dizer o que fazer. Você tem a liberdade para fazer o que melhor funcionar dentro das suas necessidades e possibilidades. Por essa flexibilidade é que dizemos que o Scrum é um framework e não uma metodologia. Assim como qualquer framework o Scrum pode ser estendido, tendo algumas de suas práticas removidas se for o caso, mas cuidado, pois a cada remoção valor esperado pode se perder durante o processo.

O Scrum é fundamentado no empirismo que afirma que o conhecimento vem da experiência e da tomada de decisões baseadas no que é conhecido. O Scrum emprega uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos. Três pilares apoiam a implementação de controle de processo empírico: **transparência, inspeção e adaptação**.

A **transparência** diz que os processos devem estar visíveis aos responsáveis pelos resultados. Para isto, faz-se necessário um padrão comum para que todos compartilhem do mesmo entendimento do que está sendo visto. Exemplo: uma definição de pronto deve ser clara e compartilhada por todos. Quando um membro do time falar que uma funcionalidade está pronta, todos saberão o que isto significa.

A **inspeção** deve ser realizada com frequência analisando os artefatos Scrum e o progresso em direção a detectar variações. As inspeções trazem benefícios quando os inspetores são especializados no trabalho de verificação

A **adaptação** ocorre quando na inspeção um ou mais aspectos de um processo ficaram além ou aquém dos limites aceitáveis ou quando o produto resultado é inaceitável. O ajuste deve ser realizado o mais breve possível para minimizar os desvios.

Toda documentação referente ao Scrum pode ser encontrada no *Scrum Guides* ou Guia do Scrum - http://www.scrumguides.org/

Figura 8 – o framework Scrum

O framework Scrum possui:

- Três papéis
- Três artesanatos
- Quatro cerimônias

Cada um dos itens será descrito abaixo.

Os papéis

Figura 9 – os papéis no Scrum e como se comunicam

Dono do produto / product owner (PO)
É responsável por gerenciar o *backlog* do produto, garantir o ROI (retorno do investimento), definir a visão do produto, gerenciar a entrada de novos requisitos e definir a prioridade e aceitar ou rejeitar o que será entregue no final de cada iteração. O dono do produto é comumente chamado de PO, que quer dizer *product owner.*

Scrum Master (SM)
É a pessoa que mais conhece Scrum dentre todos os papéis. E por conhecer a fundo o Scrum, ele auxilia o dono do produto na criação e ordenação do *backlog* do produto. Ele mantém vivo os valores que devem ser seguidos no Scrum. Ele também deve ser o facilitador nas cerimônias e também ajuda a remover os impedimentos que a equipe de desenvolvimento encontra durante a Sprint. O Scrum Master não deve ser autoritário afinal ele não é o dono do time. A forma de trabalho de um Scrum Master deve desafiar todo o time Scrum a ser mais eficiente.

O Scrum Master tem a responsabilidade de fazer com que as cerimônias fluam de forma adequada, utilizando técnicas de facilitação. Além disto, o controle do tempo das reuniões (*timebox*). Por exemplo, na reunião diária (*daily meeting*) não é função conduzir a mesma.

Equipe de Desenvolvimento
É uma equipe responsável por desenvolver incrementos no produto. Também é responsável pela estimativa quanto ao tamanho dos itens do *backlog*. A equipe deve ser auto-organizada e nem muito grande. O *Scrum Guide* sugere times 7 pessoas. Mike Cohn, conta em seu livro *Succeeding with Agile,* como a Amazon monta os seus times de Scrum. A Amazon sugere o tema **"times de 2 pizzas"** por afirmar que o time deve ter o número de pessoas que possa ser alimentado com apenas duas pizzas. Esta é uma boa referência na hora de trabalhar com Scrum. Os benefícios de ter um **time de 2 pizzas** são:

- Menos Ociosidade Social: Ociosidade Social é a tendência das pessoas se esforçarem menos do que são capazes por acharem que haverá outras pessoas que irão fazer aquele trabalho.
- Aumento da satisfação pessoal
- Especialização excessiva é menos provável de acontecer
- Menos tempo é gasto para coordenar esforços
- Interação construtiva: time com um sentimento de confiança, responsabilidade mútua e coesão.

Os artefatos
Um artefato dá uma visão do andamento do projeto e das sprints. Os artefatos são divididos em:

- Backlog do produto */ product backlog*
- Backlog da Sprint */ sprint backlog*
- Incremento do produto

Um sprint é um tempo definido de trabalho, podendo ser de:

- 1 semana de trabalho,

- 2 semanas de trabalho
- 4 semanas de trabalho.

Em cada sprint são realizados as cerimônias que serão descritas logo mais a frente.

Backlog do produto / product backlog
É uma lista ordenada de funcionalidades que precisam ser desenvolvidas. A criação, remoção e reordenação são de responsabilidade do dono do produto (PO). O Scrum Master auxilia o dono do produto nesta tarefa.

Cada funcionalidade na lista é denominada história do usuário. As mais importantes ficam no topo da lista e serão implementadas primeiro.

Para determina a história mais importante leva-se em conta o conhecimento daquela funcionalidade.

O *backlog* do produto pode conter itens funcionais, não funcionais, arquiteturais e de infraestrutura. Pode conter também bugs, itens para refatoração e itens que representem riscos a serem removidos.

BACKLOG DO PRODUTO		
História do usuário	**Prioridade do negócio**	**Pontuação**
Tela de Cadastro de Veículos	1	2
Relatório Curva ABC	2	8
Otimização da consulta a tabela Pessoa	3	5

Com a ordenação do *backlog* definida, faz-se necessário a estimativa do mesmo. Várias técnicas podem ser utilizadas. A mais comum é a técnica do *planning poker*. O mais importante é entender que quem pode dizer o tamanho que é cada história não é o dono do produto e sim a equipe de desenvolvimento.

O *planning poker* funciona da seguinte maneira:

1. A equipe de desenvolvimento já possui uma referência definida para a carta coom valor 2. Por exemplo: 2 representa uma tela de cadastro com as opções de listagem, inclusão, remoção e alteração, sem relatórios. Ou seja, um CRUD clássico (CRUD é o acrônimo da expressão do idioma inglês, *Create* (Criação), *Retrieve* (Consulta), *Update* (Atualização) e *Delete* (remoção))
2. O Scrum Master vai até a equipe de desenvolvimento já com o *backlog* do produto priorizado
3. O Scrum Master lê a tarefa em voz alta e responde as dúvidas da equipe de desenvolvimento
4. Cada desenvolvedor escolhe a carta em segredo e após o comando, todos apresentam suas cartas ao mesmo tempo.
5. Os valores são comparados. Caso haja divergência, o Scrum Master pergunta aos de maior e de menor pontuação o porquê de suas escolhas. Neste momento pode-se perceber que os que escolheram cartas muito baixas, se

comparada com os demais, não entenderam realmente a tarefa. E os que escolheram cartas muito altas, talvez estejam enxergando uma complexidade não declarada.

6. O processo de votação se repete até que um consenso exista.

Figura 10 – cartas do *planning poker*

A pontuação das cartas é inspirada na sequência de Fibonacci, com a adição de algumas cartas:

- Valor 0: não precisa ser feito, provavelmente já foi implementada ou tem uma forma similar no sistema que pode atender a necessidade do PO.
- Valor 20 ou 40: tarefa grande demais para ser estimada. A sugestão é quebra-la em itens menores
- Valor 100: tarefa muito grande e não é possível fazer em *sprints*. Time sugere que seja quebrada.
- Valor ?: Não ficou claro o que precisa ser feito.

Em alguns cenários, a carta com uma xicara de café também está presente. Nela o time sugere uma pausa antes de escolher uma carta com valor.

Figura 11 - a carta café

Backlog do Sprint / Sprint backlog

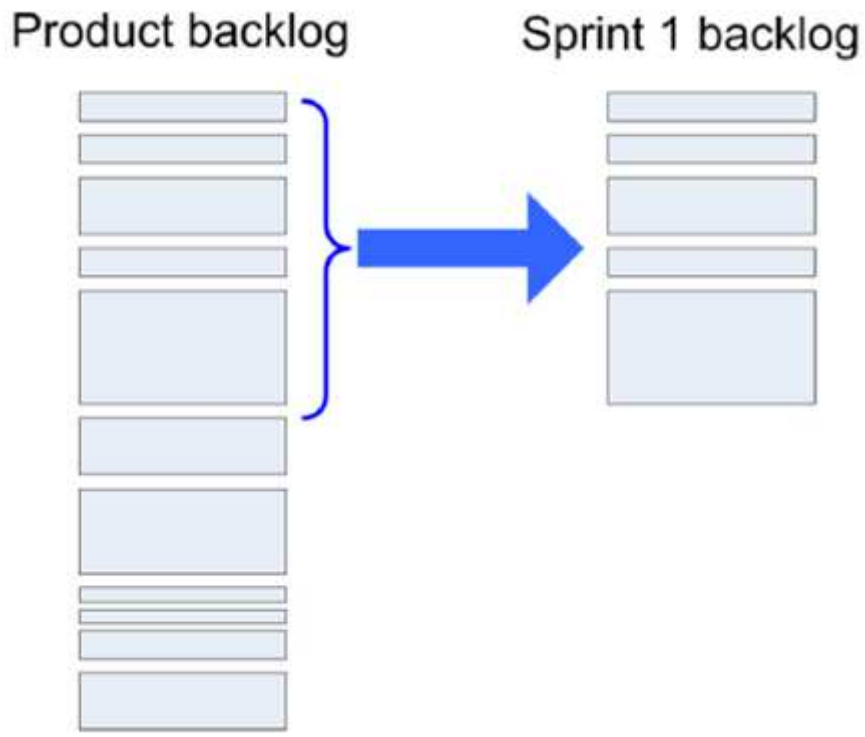


Figura 12 – criação do backlog do sprint

O *backlog* do *sprint* é o conjunto de itens que foram selecionados para serem implementados durante a Sprint. Um backlog da Sprint é resultado esperado após a reunião de planejamento da Sprint (*Sprint* Planning). Ele deve ter uma meta, os itens selecionados e as tarefas técnicas necessárias para transformar o item em um incremento do produto.

Quem adiciona itens no backlog da Sprint é a equipe de desenvolvimento.

BACKLOG DA SPRINT		
META DA SPRINT: USUÁRIOS PODERÃO CADASTRAR VEÍCULOS		
História do Usuário	**Demanda técnica**	
Autenticação no sistema	Criação da tela de login e senha	Permitir autenticar com a conta Facebook
	Criação das tabelas na base de dados	Permitir autenticar com a conta Twitter
	Permitir autenticar com a conta Google	
Cadastro de veículos	Criação das tabelas na base de dados	
	Criação da tela de veículos	

Incremento do produto

O incremento é o resultado final de cada *sprint*. Ao realizar a entrega do incremento, o dono do produto vai perceber o valor do investimento e também vislumbrar outras possibilidades. A equipe de desenvolvimento deve entender que o incremento é algo

potencialmente entregável. Para saber se é algo entregável, o time deve entender que o que foi entregue pode ir imediatamente entrar em funcionamento (ser disponibilizado em produção).

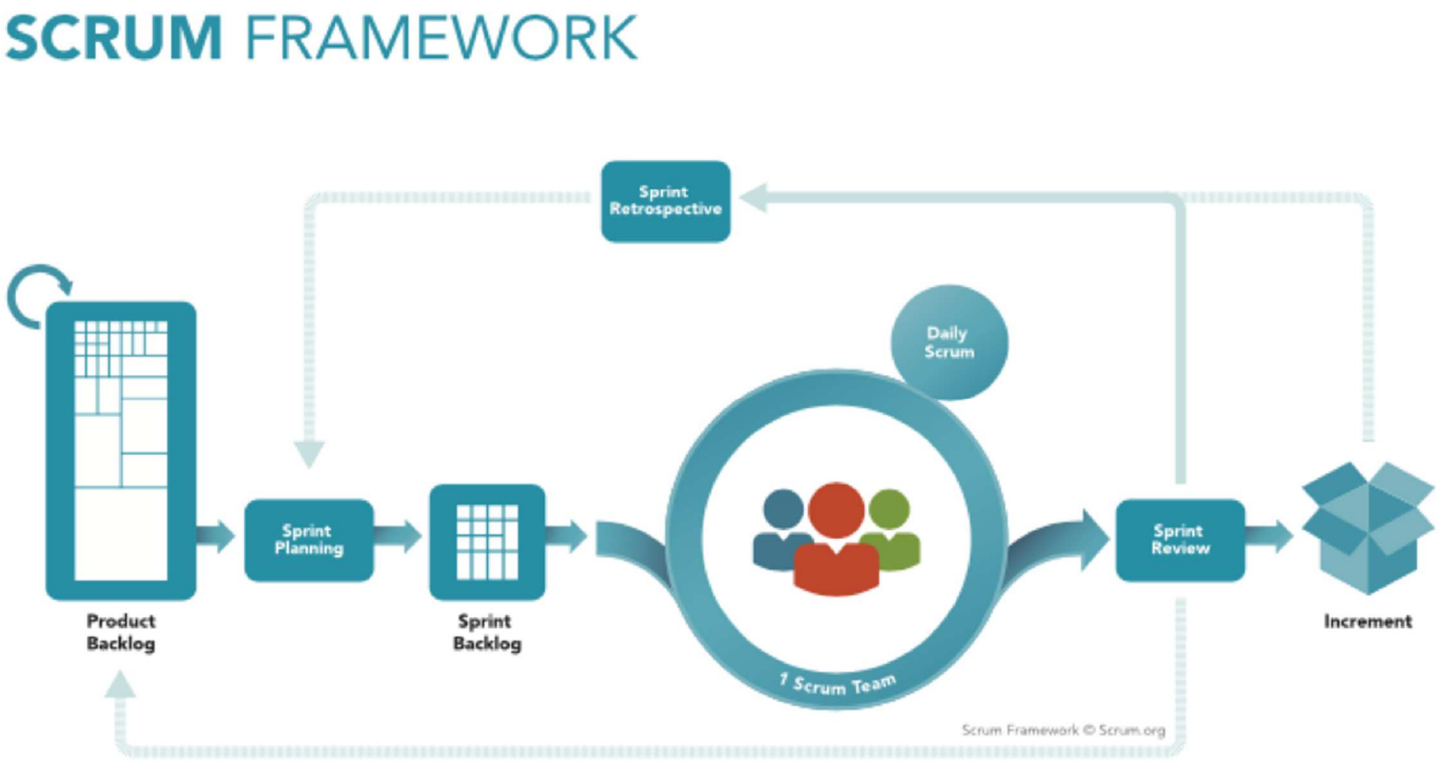


Figura 13 - o framework scrum

As cerimônias

As cerimônias são eventos de duração fixa (*timebox*). Cada cerimônia é uma oportunidade para inspeção e adaptação e deve ocorrer em intervalos regulares. Abaixo vamos detalhar cada cerimônia do Scrum.

Sprint

É um ciclo completo de desenvolvimento de sistema com duração fixa de tempo. Aconselha-se a utilização de intervalos de 2 semanas de duração. Utilizar mais que 4 semanas não é aconselhável. Ao final da sprint temos um **incremento do produto**. A cada final de Sprint temos o *feedback* do dono do produto quanto ao que está sendo desenvolvido. A Sprint é formada por quatro cerimônias:

- Reunião de planejamento
- Reuniões diárias
- Reunião de revisão
- Reunião de retrospectiva

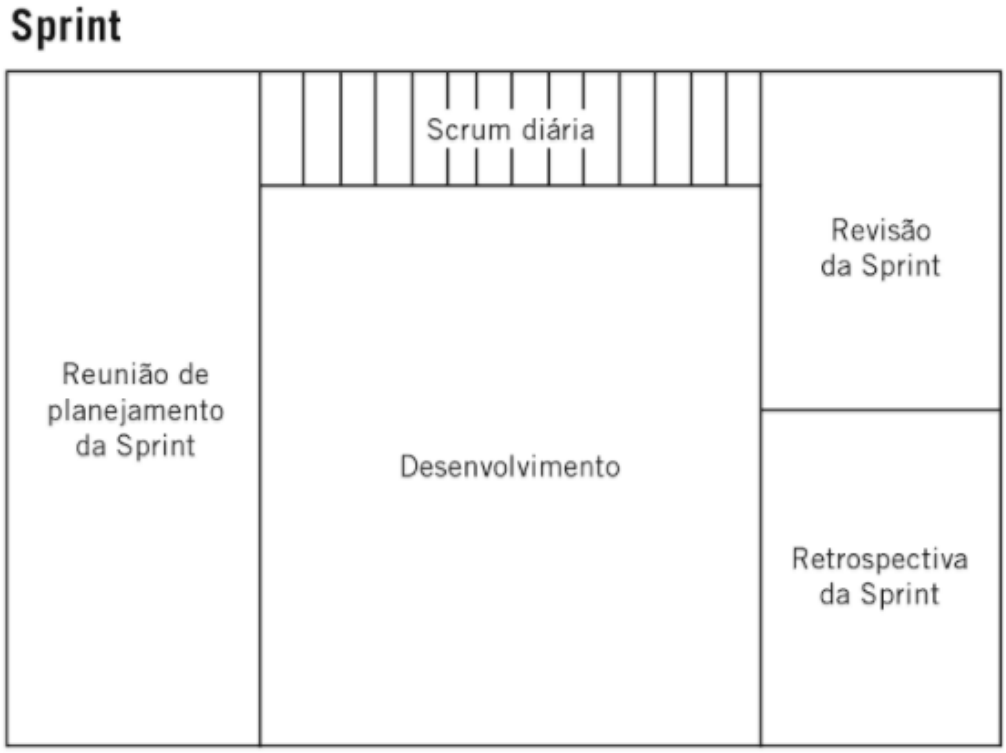


Figura 14 - cerimônias da sprint

Reunião de planejamento da Sprint / planning meeting / Sprint planning
A reunião de planejamento também é chamada de *planning meeting* ou *sprint planning*. Ela acontece no início de cada sprint. Ela tem uma duração (*timebox*) de 8h para sprints de 1 mês (4 semanas). Para sprints menores, sugere-se o uso da proporcionalidade de semanas X horas.

Ela é dividida em:

- O que será entregue como incremento de produto nesta sprint?
- Como faremos para entregar o incremento?

O dono do produto deve apresentar os itens que estão no topo do backlog e que estão estimados. Cabe a equipe de desenvolvimento informar o quanto de itens serão atendidos na sprint.

O resultado desta reunião é o *backlog* da sprint, relatado anteriormente.

Reunião Diária / Daily Meeting /Daily Scrum
A reunião diária também é chamada de d*aily meeting* ou d*aily scrum*. Ela ocorre diariamente e tem tempo máximo de 15 minutos. Ela é uma reunião para a equipe de desenvolvimento. Cada membro da equipe de desenvolvimento deve responder as seguintes perguntas aos outros membros:

- O que fiz desde a última reunião diária?
- O que pretendo fazer até a próxima reunião diária?
- Existe algo me impedindo de concluir alguma tarefa?

Ela deve ser realizada em pé. Isto irá garantir que o *timebox* da reunião seja cumprido. Esta reunião visa melhorar e sincronizar a comunicação da equipe de desenvolvimento. Ela não deve ter o intuito de reportar status das tarefas, pois isto é feito pelo quadro de tarefas. Caso algum membro tenha algum impedimento, a própria equipe se auto-organiza para solução deste. O que a equipe não consegue resolver, é classificado como **impedimento** e informado o scrum master para que este encontre a melhor maneira de remover o impedimento.

Figura 15 – a reunião diária deve ser em pé

Revisão da Sprint / Sprint Review

Esta reunião ocorre no final da sprint. Os participantes desta cerimônia são a equipe de desenvolvimento, scrum master e o dono do produto. Caso haja necessidade, outras pessoas podem ser convidadas.

O objetivo desta cerimônia é apresentar o que foi desenvolvido na sprint. Além disto serve para a equipe de desenvolvimento colher opiniões e impressões dos presentes para adaptar-se para a próxima sprint. O foco é o aprimoramento do produto.

Nesta reunião, a meta da Sprint é validada e o progresso atual do projeto é atualizado.

Retrospectiva / Retrospective

Esta reunião não é menos importante que as demais. Alguns autores a consideram a mais importante, pois é o momento que toda equipe scrum (dono do produto, scrum naster e equipe de desenvolvimento) aprimoram o processo. Nesta reunião são levantados tudo o que funcionou e o que precisa ser melhorado. Para que isto ocorra, o scrum master utiliza-se de diversas técnicas e dinâmicas para o bom andamento da mesma.

Deve-se existir um cuidado nesta reunião para que a mesma não seja nociva aos membros da equipe scrum, pois em alguns momentos a “roupa suja” será lavada. Uma forma de gerar equilíbrio na reunião é compartilhar a visão que se espera ao participar

da mesma. Pensando nisto KERTH,2001 definiu o que é chamada de **diretiva primária**:

"Independentemente do que descobrimos, nós entendemos e realmente acreditamos que todos fizeram o melhor trabalho que poderiam, dado o que era conhecido na época, suas habilidades e competências, os recursos disponíveis, bem como a situação em questão."

Com esta declaração, conduzimos a equipe scrum para uma mentalidade colaborativa e podemos começar a reunião.

DERBY,2006 categorizaram uma retrospectiva em cinco momentos:

- **Abertura** – o momento para se estabelecer um ambiente que favoreça as pessoas a falaram de forma franca e aberta, revisar os objetivos da reunião. Neste momento a diretiva primária deve ser apresentada;
- **Colher dados** – momento para trazer as perspectivas de cada membro do time sobre a sprint que se passou criando uma imagem compartilhada para todos;
- **Gerar ideias** – aqui se discute sobre os assuntos levantados no momento anterior, se aprofundando nos assuntos através de atividades que auxiliam esse fim;
- **Decidir o que fazer** – finalmente se elencam pequenas melhorias que farão diferença para o time;
- **Fechamento** – Se resume como o time seguirá o plano e os comprometimentos. Este é o momento de agradecer o time e colher feedbacks e percepções de como foi a retrospectiva.

Alguns sites como o Retro ágil e o Fun Retrospectives ajudam o Scrum Master a planejar as dinâmicas e a sequência da reunião. Para conhecer mais sobre estes materiais, acesse:

- Retro ágil: https://retroagil.wordpress.com/
- Fun Retrospectives: http://www.funretrospectives.com/

Gráfico de burndown

O gráfico de *burndown* demonstra o progresso do time durante uma Sprint. Ele possui a quantidade de trabalho a ser feita no eixo Y com a quantidade de dias da Sprint (eixo X). Se você esta utilizando pontuação do *planning poker*, você deve somá-los e terá o a quantidade de trabalho a ser feita (eixo Y). O eixo X representa os dias ou meses, de projeto, mas desaconselha-usar exibir sábado e domingo no gráfico por não serem dias anterior acesso, em iterações de uma a Nallva

No centro do gráfico, vamos ter uma linha central que descreve o andamento esperado dos incrementos. No exemplo da figura 16, esta linha é a vermelha ou a linha que não osciliou

Já a outra linha do gráfico representa a quantidade de entregas num dia. O ideal é que os dois gráficos caminhem o mais próximo possível.

Figura 16 - Gráfico Burndown

Através deste gráfico, você poderá encontrar respostas para as seguintes perguntas:

- Quão bom é esse time no planejamento?
- O time é realmente auto-organizado e está trabalhando como uma equipe?
- Quais melhorias esse time pode fazer?
- O quão bem o time está executando as histórias planejadas no Sprint?

Este gráfico pode ser usado em outras formas de agilidade;

Para finalizarmos, vamos fazer uma revisão sobre o que vimos referente ao Scrum:

- É formado por equipes pequenas e multidisciplinar
- Produção de versões incrementais de um software em iterações curtas
- Equipes se auto-organizam para planejar e desenvolver o trabalho nos seus sprints
- Trabalho diluído em cada integrante da equipe
- Trabalho em equipe é facilitado pelo scrum master
- Scrum master não trabalha diretamente com atividades técnicas, mas remove impedimentos e reforça os pontos centrais do Scrum ao longo do desenvolvimento do produto.
- Trabalho é organizado a partir do *backlog* do produto
- O *backlog* do produto deve ser revisado e priorizado constantemente
- A comunicação e cooperação entre as equipes se intensificam ao longo das *sprints.*

Capitulo 3 - Existe vida além do Scrum

Quando se fala em métodos ágeis, muito se fala do Scrum. Mas Scrum não pode ser considerado **bala de prata**, ou seja, algo que resolve todos os problemas. No framework Scrum, o trabalho é "puxado", sendo dirigido por cerimônias bem definidas. O Scrum pressupõe a possibilidade de planejamento e execução com meta e finalidade. Mas este cenário nem sempre atende todas as demandas. Numa equipe de manutenção, sustentação ou infraestrutura, a previsibilidade das tarefas é reduzida. Para isto, o Kanban se encaixa melhor.

À medida que o ambiente está estabilizado, um novo trabalho pode começar a ser feito que seja o de enxugamento, otimização e geração máxima de valor. Para isto temos o Lean, que é uma derivação do *Lean Manufactury*, já conhecido da empresa Toyota.

Além do Kanban e Lean, temos outros métodos ágeis que são:

- FDD – *Feature Driven Development*
- OpenUP

Vamos detalhar cada um deles logo a seguir.

FDD – Featrue Driven Development

Representa o desenvolvimento por *feature* ou funcionalidade pequena. Esta funcionalidade deve ter valor claro para o cliente. Entende-se que uma funcionalidade deva durar no máximo 80h de desenvolvimento.

Uma funcionalidade deve ser descrita com a seguinte nomenclatura:

[AÇÃO] [RESULTADO][OBJETO]

Exemplos:

- Enviar nota fiscal para SEFAZ
- Imprimir boletos parcelados para um cliente

Esta convenção permite padronização na identificação das funcionalidades e aperfeiçoa a comunicação. Além disto, fornece fortes dicas para a codificação correta de classes, métodos e parâmetros, permitindo assim uma linguagem semelhante entre o negócio e a equipe técnica.

FDD é composto por **cinco processos**, tendo como matéria-prima os requisitos. Estes cinco processos estarão categorizados em duas fases:

- **Inicialização**: é uma fase linear que tem o objetivo de criar um plano inicial de entregas incrementais. Ela costuma durar de 2 a 4 semanas.
- **Construção**: é uma fase iterativa a qual tem como meta a entrega de incrementos ao produto de forma frequente, funcional e com qualidade para ser utilizado pelo cliente. Tradicionalmente, as iterações são de 2 semanas.

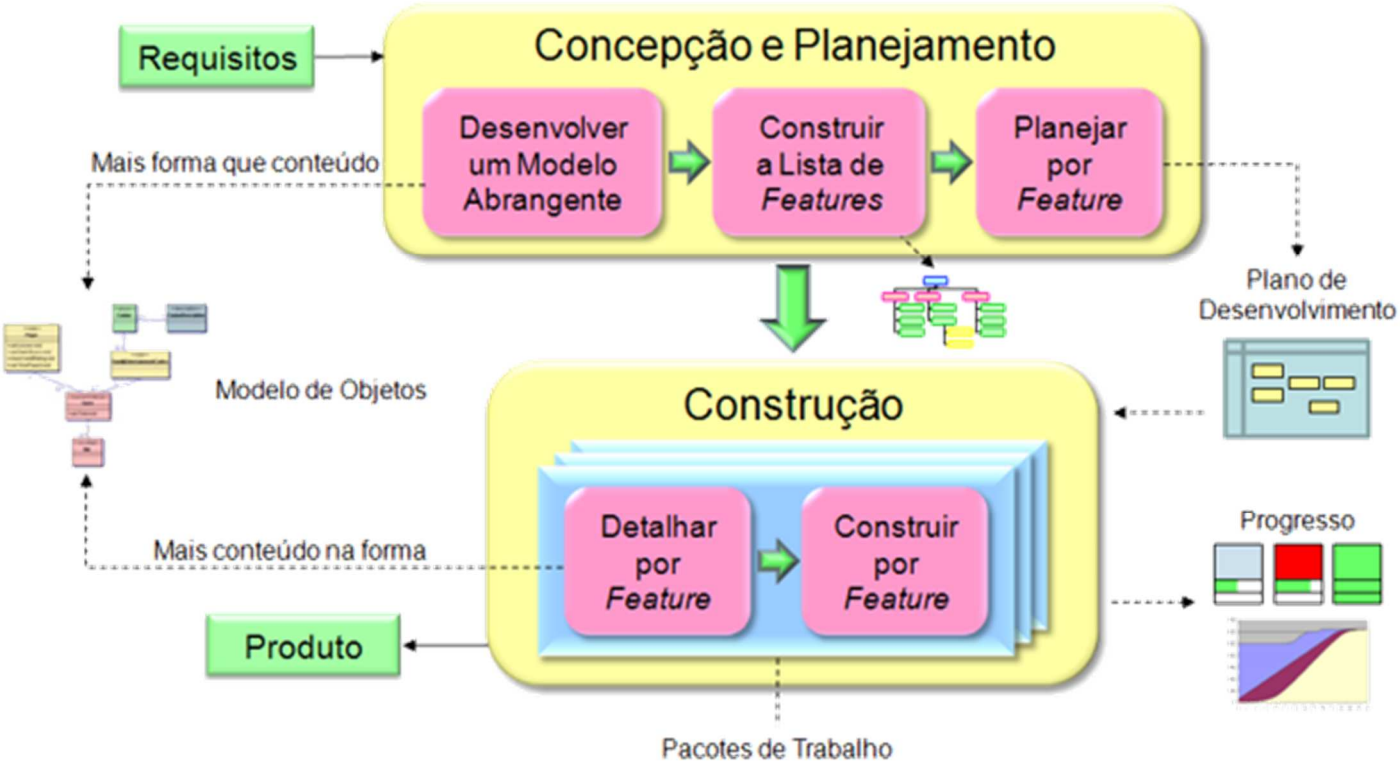


Figura 17 – o quadro de processos do FDD

Os cinco processos são:

DMA – Desenvolver um modelo abrangente
Realizada por membros do domínio do negócio e por desenvolvedores. Neste momento busca-se o compartilhamento do conhecimento para tornar os requisitos mais claros. Busca-se uma modelagem suficiente para iniciar o desenvolvimento. Não se espera toda a modelagem como é realizado no modelo em cascata.

CLF – Construir a lista de funcionalidades
Nesta etapa são identificadas as principais funcionalidades que satisfaçam os requisitos. Para facilitar a identificação, usa-se a categorização em três níveis:

- Áreas de negócios
- Atividades de negócios
- Passos da atividade de negócios, sendo esta a funcionalidade propriamente dita.

Exemplo:

- **Área de negócio**: Gestão de vendas
- **Atividade de negócio**: [VERBO]... ou [SUBSTANTIVO]...
  - Repor estoque
  - Entrada de mercadoria
- **Passo da atividade de negócio**: [AÇÃO] [RESULTADO][OBJETO]
  - Calcular quantidade disponível de um produto

PPF – Planejar por funcionalidade
A partir deste processo será possível ter o plano de desenvolvimento. Geralmente este plano é feito em nível de atividade de negócios, definindo o mês que se espera ter a funcionalidade implementada.

DPF – Detalhar por funcionalidade
Deve ser realizada em cada pacote de funcionalidade para que exista material suficiente para implementação. O resultado deste detalhamento serão pacotes para serem implementados pelos desenvolvedores.

CPF – Construir por Funcionalidade
É a atividade de produzir algo com valor para o cliente.

OpenUP

É um processo enxuto, baseado no *Unified Process* (UP). Para aqueles que usavam *Rational Unified Process* (RUP) não irão sentir tanta dificuldade ao decidirem utilizar o OpenUP. O ciclo de vida é interativo e incremental. Enquanto o RUP é um framework muito completo (atendendo quase todas as possibilidades), o UP é um processo de engenharia de software mínimo e que pode ser customizado.

Princípios do OpenUP
São quatro princípios que devem ser seguidos. São eles:

- Balancear prioridades competidoras para maximizar o valor aos envolvidos no projeto
- Colaborador para alinhar interesses e compartilhar entendimento
- Focar cedo na arquitetura para minimizar riscos e organizar o desenvolvimento
- Evoluir para continuamente obter *feedback* e melhorar
- Representar o esforço, em horas ou dias, de um grupo de até 3 pessoas.

O processo é dividido em 3 camadas distintas:

Micro incremento
- É composto por item de trabalho de uma iteração
- Estas pessoas interagem para atingir um objetivo definido.

Ciclo de vida da iteração
- É composto por um plano de iteração.
- Cada iteração é focada numa versão/build
- Esta versão será entregue ao cliente em algumas semanas
- Em cada iteração existe a aplicação dos passos: requisitos, arquitetura, implementação, teste e gerência de projetos

Ciclo de vida do projeto
- É composto pelo processo como um todo.
- Dividido em iniciação, elaboração, construção e transição
- Cada etapa é encerrada com um marco, ou seja, um conjunto de atividades e/ou artefatos gerados pela equipe de desenvolvimento.

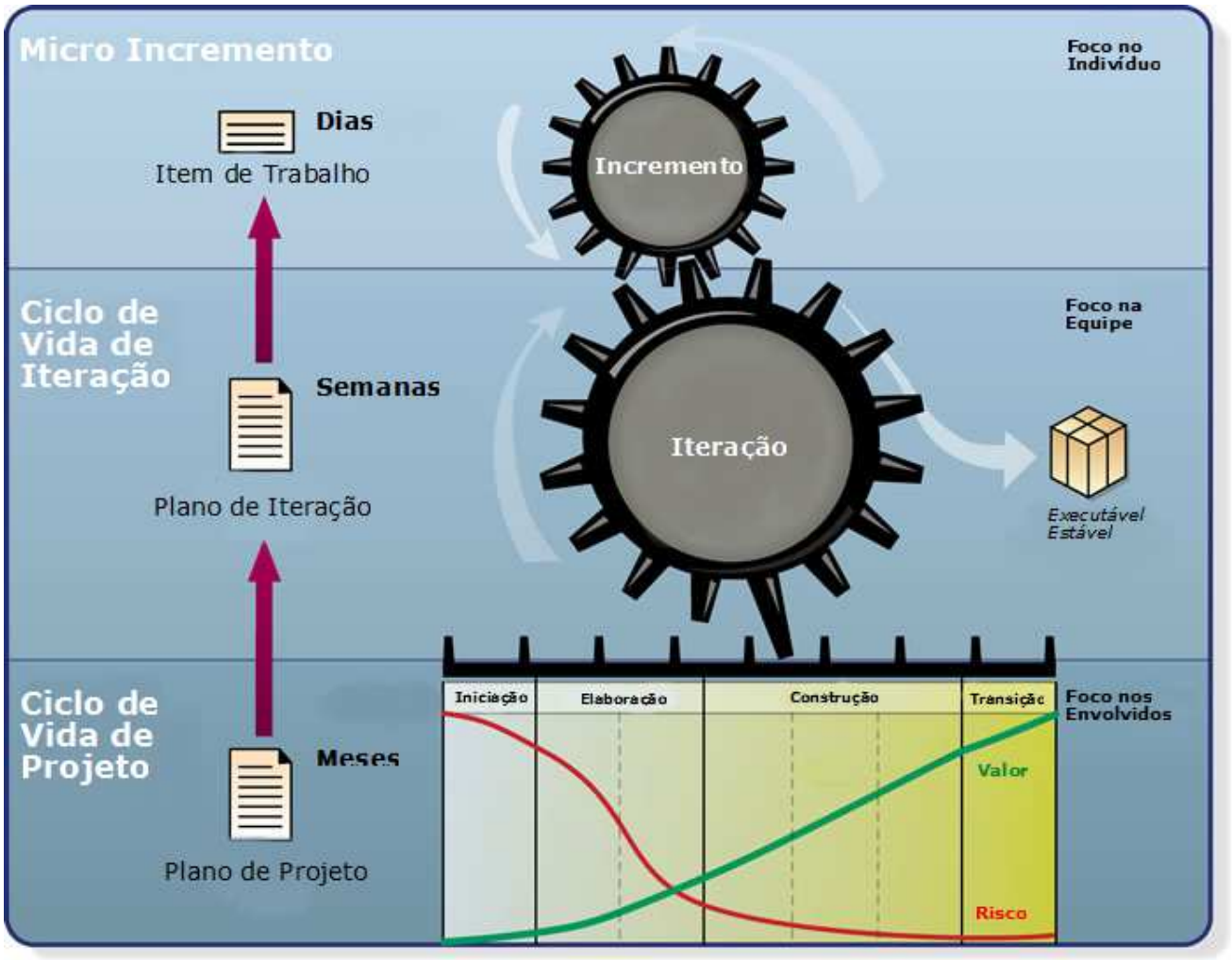


Figura 18 - OpenUP

Kanban

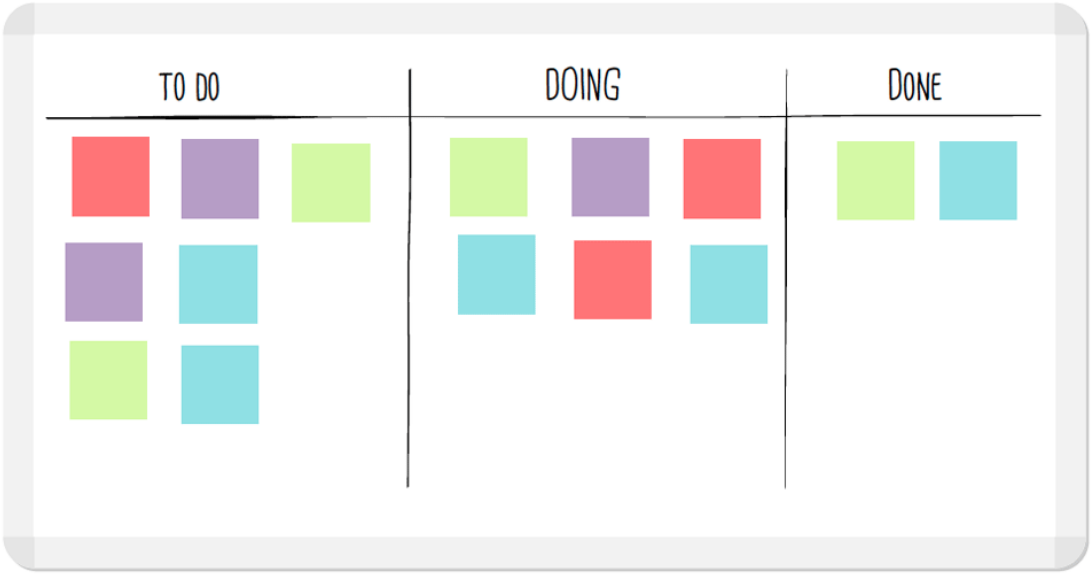


Figura 19 – um quadro básico do Kanban

Até o momento estudamos Scrum, FDD e OpenUP e podemos perceber elementos comuns entre eles: o **valor** e o **fluxo**.

Kanban se utiliza desses elementos comuns a maioria dos processos para projetar um mapa visual que represente o modelo de trabalho na forma como ele é. Este mapa visual é semelhante ao quadro da figura 18. Com Kanban, é possível enxergar os gargalos que interferem no fluxo de valor. Para resolver o gargalo, o time deverá se reorganizar e com isto o processo muda e o mapa visual também. Com a mudança, uma nova forma de ver o trabalho é evidenciada e novas oportunidades de melhoria emergem. Um ciclo evolucionário é estabelecido.

Para a adoção do Kanban, duas premissas devem ser seguidas:

- O trabalho deve ser dividido em pequenos incrementos que adicionarão alguma forma de valor ao cliente final. E estes pequenos incrementos podem ser gerenciados de forma independente
- Os pequenos incrementos devem ser desenvolvidos em um fluxo contínuo numa cadeira de valor desde seu início até o fim (entrega do produto).

O Kanban se apoia fortemente na gestão visual. É através dela que o andamento dos trabalhos é evidenciado e decisões de mudança de percurso são identificadas. Para isto utilizamos de *flipcharts*, quadros, post-it, adesivos personalizados ou então ferramentas online, como o *Trello*. Mas para manter a gestão visual quando se usa ferramentas online, faz-se necessário deixar o quadro de trabalho sempre visível através de um monitor visível a todos.

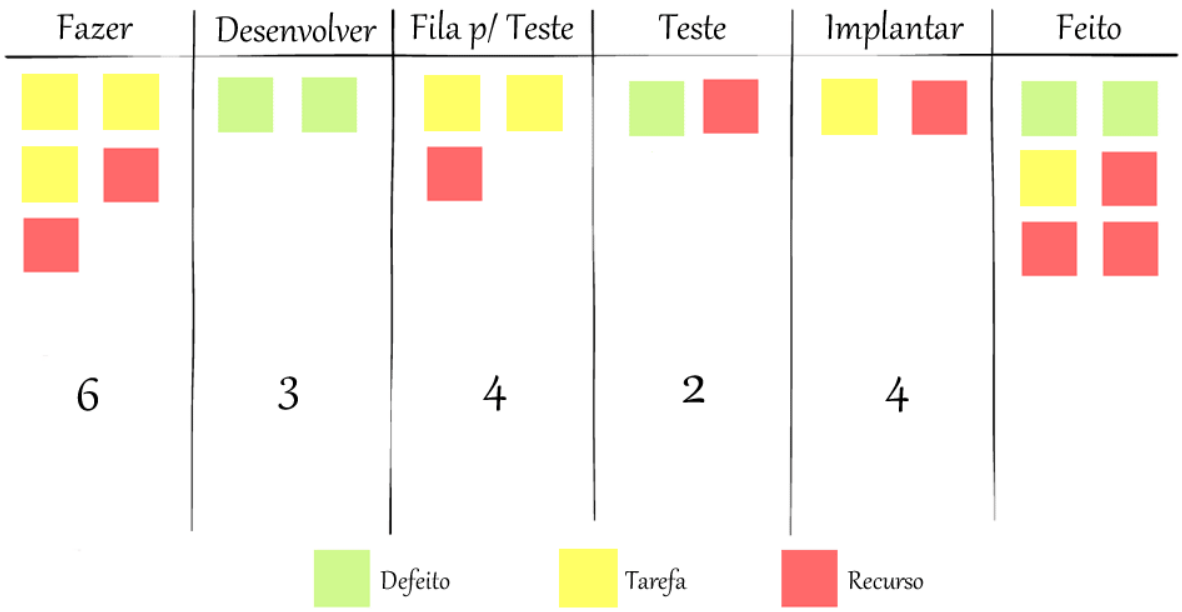


Figura 20 - Quadro Kanban com tarefas categorizadas

Além da visibilidade, o Kanban trabalha com o que chamamos de fluxo puxado. As atividades em progresso são limitadas (WIP – *work in progress*). Itens em progresso devem sair antes que novos itens entrem. O processo se torna puxado pela abertura de capacidade, ao invés de empurrado, o que ocorre quando a capacidade é ignorada.

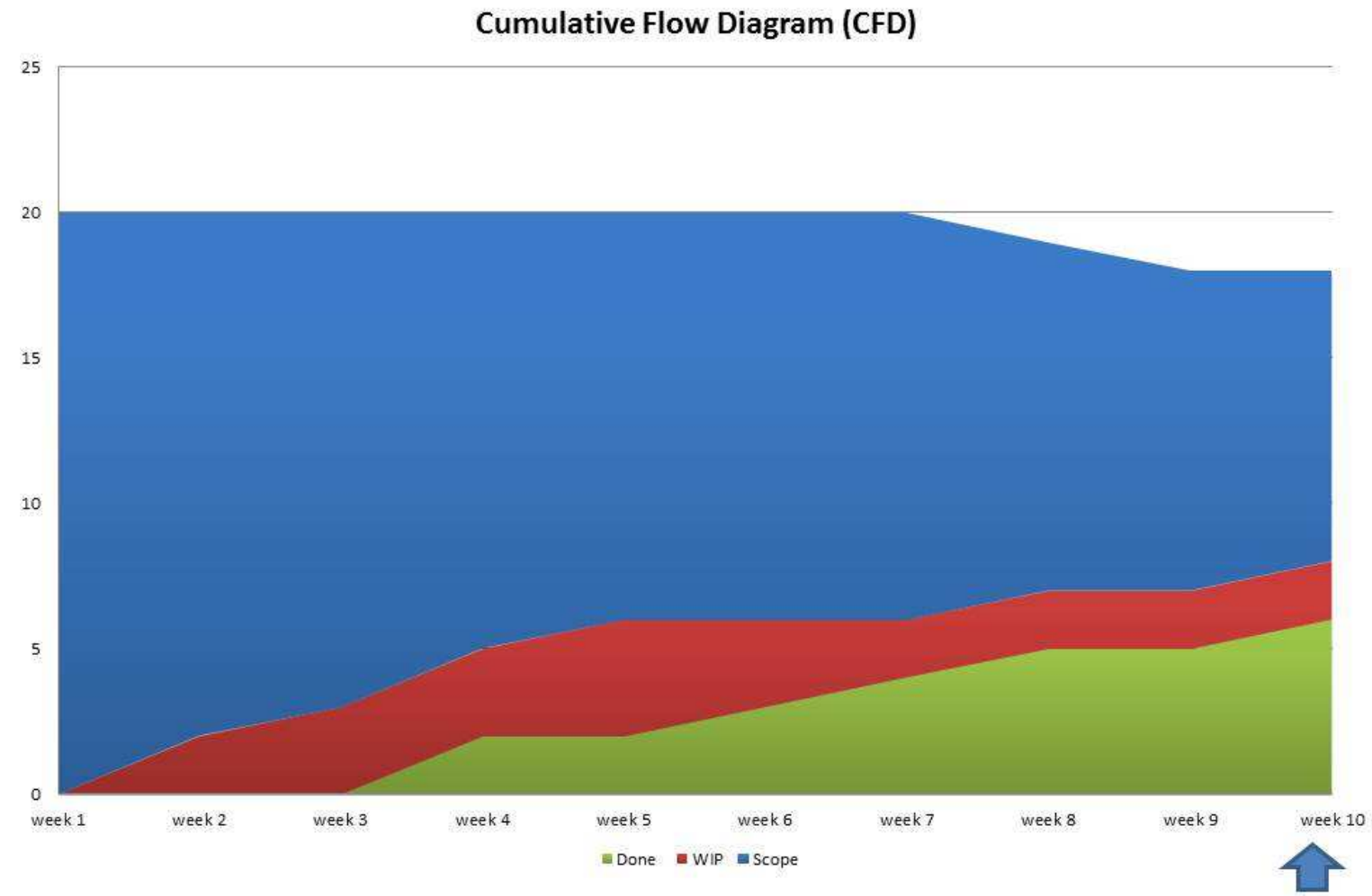
Outra característica do Kanban é a busca pela melhoria contínua: O mapa de trabalho aliado às conversações frequentes faz com que o ambiente de trabalho esteja mais suscetível a mudanças e a experimentações de novas formas de se trabalhar. Esses experimentos funcionam como uma espécie de "seleção natural", onde as práticas, regras e comportamentos que funcionam são absorvidas, enquanto aquelas que se provam ineficientes são eliminadas.

E por fim, as métricas estão presentes e nos ajudam a ter números referente a melhoria contínua. As métricas são fundamentais para que seja possível avaliar se uma mudança levou a uma melhora ou não em todo contexto. Aqui vale a máxima: "Não se pode melhorar o que não se pode medir". Sendo assim, em projetos Kanban devem ser adotadas métricas para avaliar a evolução do modelo. Vale lembrar que estas métricas podem ser utilizadas em outras metodologias e frameworks.

São exemplos de métricas:

- Diagrama de fluxo cumulativo (*Cumulative Flow Diagram*):
- *Lead time*
- Tempo de ciclo (*cycle time*)
- *Throughput*
- *Índices de defeitos*

Diagrama de fluxo cumulativo (*Cumulative Flow Diagram*)
Ele apresenta a quantidade de itens em cada situação do fluxo ao longo do tempo. Ele pode substituir o gráfico de *burndown.* Sua atualização é fácil e fornecem dados do projeto como um todo. Ele possui os dados do *burndown* e mais alguns.



time	Scope	WIP	Done
week 1	20	0	0
week 2	18	2	0
week 3	17	3	0
week 4	15	3	2
week 5	14	4	2
week 6	14	3	3
week 7	14	2	4
week 8	12	2	5
week 9	11	2	5
week 10	10	2	6

Figura 21 – Fluxo cumulativo no Kanban

A área em verde, que representa as demandas finalizadas (*done*), representa a velocidade no decorrer da iteração.

A linha em vermelho é denominada WIP (*work-in-progress*) e a linha azul vai representar o *backlog*.

Entender o porquê destas linhas é importante para identificação de melhorias. Alguns pontos que você deve monitorar:

- Se a distância entre as duas linhas no WIP aumentar, isto pode ser sinal de gargalo.
- Se a linha do *backlog* estiver mais inclinada do que a linha *Done*, identificam-se a inclusão de trabalho acima da capacidade suportada pelo time.
- Para estimar a data final da entrega, projeta-se um cenário aonde as linhas de *backlog* e *done* irão se encontrar.
- O tempo médio de ciclo e quantidade de item na fila, já são oposições.

Lead time

Lead time é o tempo, geralmente em dias, desde o início de uma história (a criação da demanda) até a sua entrega (finalização ou implantação).

Ele engloba o tempo gasto com o desenvolvimento da demanda. Caso você deseja saber somente o tempo de desenvolvimento da tarefa, este é chamado de templo de ciclo (cycle time).

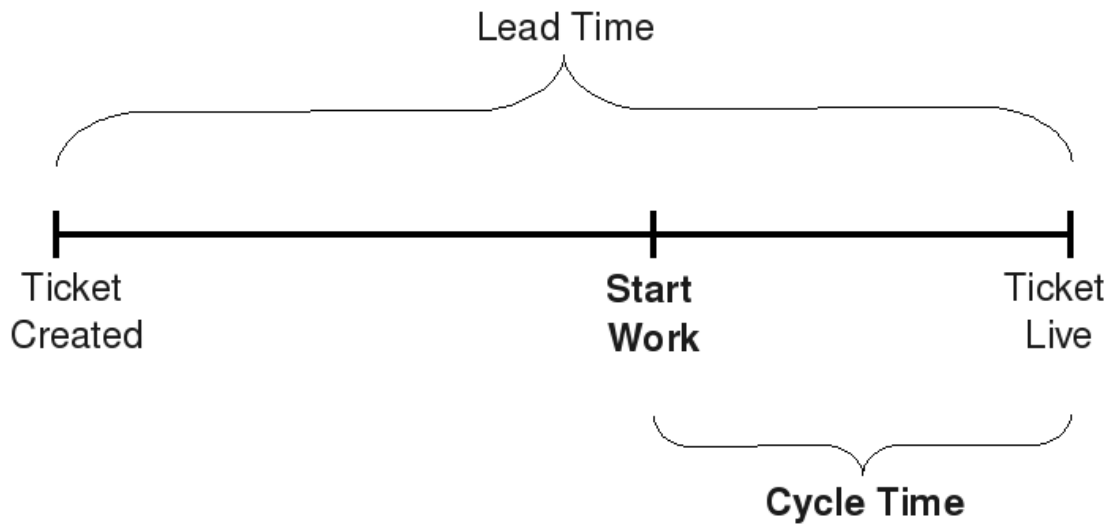


Figura 22 - Diferença de *Lead Time* e *Cycle Time*

Para elaborar este indicador, é necessário preencher a data da solicitação/criação da demanda e a data que houve a finalização da mesma. O resultado destes dados gerará um gráfico semelhante ao que vemos em sequência.

Figura 23 - Gráfico de *lead time*

O único problema deste gráfico é que você tem que trabalhar com tarefa já concluídas. Estes dados geram informações importantes, mas só no final da iteração.

Tempo de ciclo

Este indicador também é chamado de *Cycle Time* e ele está dentro do *Lead Time*. Enquanto o *lead time* verifica tempo total da demanda, o templo de ciclo verifica somente o tempo investido no desenvolvimento da demanda. A diferença deles pode ser conferida na figura 21.

Utilizá-lo para medir os ciclos individuais irá ajuda-lo a elaborar a previsibilidade de entregas de desenvolvimento. Por isto é considerado uma métrica chave.

Para elaborar este indicador, é necessário preencher a data de inicio que se começou a trabalhar na atividade e marcar o número de dias gastos para finalizar o item. Para isto podemos usa a formula da figura 23.

$$\text{Cycle Time} = \frac{\text{Work in Progress}}{\text{Throughput}}$$

Figura 24 - Fórmula Templo de Ciclo

O resultado destes dados gerará um gráfico semelhante ao que vemos em sequência. Neste gráfico, a linha em vermelho representa o tempo médio de ciclo.

Figura 25 - Linha vermelha representa o Cycle Time

Este indicador tem o mesmo problema do *lead time*. Ele só poderá ser utilizado no final da produção da demanda.

Throughput:
É quantidade de tarefas entregues em um determinado período de tempo. *Throughput* significa vazão e para determina-lo: usamos a fórmula apresentada na imagem 23, mas de maneira adaptada.

$$\text{Throughput} = \frac{\text{Work in Progress}}{\text{Cycle Time}}$$

Figura 26 - Fórmula de cálculo - Throughput

Este indicador irá responder algumas perguntas, em especial:

- Qual é a % de itens levam menos de uma semana no gráfico? Este numero pode representar o quanto o time consegue atender semanalmente.

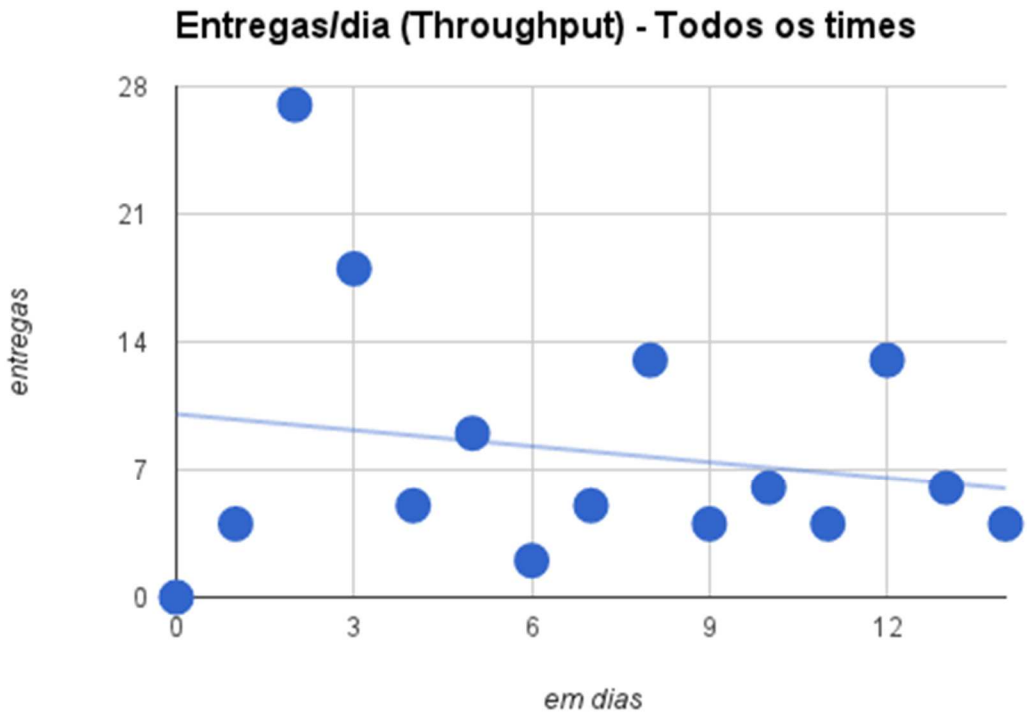


Figura 27 – Gráfico de Throughput

Índice de defeitos

Este indicador pode nos ajudar a responder as seguintes perguntas:

- Os defeitos estão aumentando? Será que houve relaxamento na produção com qualidade?
- Existe relação do baixo tempo de um ciclo com a quantidade de erros?

O indicador deve exibir as semanas de trabalho no eixo X e com a quantidade de novos erros no eixo Y. Deverá haver uma linha representando a quantidade de erros abertos e uma segunda linha com os novos erros.

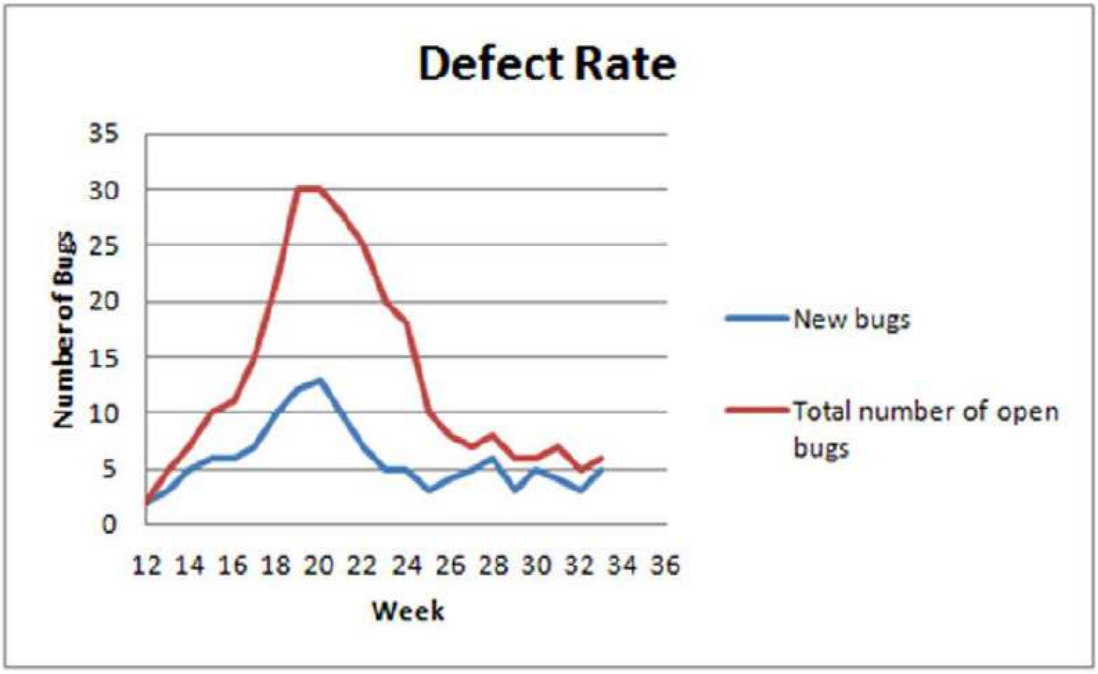


Figura 28 - Relatório de Índice de erros

Scrum e Kanban, por que não usá-los simultaneamente?!

Não há nada de ruim em juntar estas duas práticas ágeis e gerar um *"scrumban".* Mas para um time que não tem maturidade ou tempo em agilidade, aconselha-se o uso de uma ou outra ferramenta.

Scrum e Kanban são bem distintos. Vamos fazer uma comparação:

	SCRUM	KANBAN
Ritmo	Sprints regulares com extensão fixa (2 semanas)	Fluxo contínuo
Metodologia da versão	No final de cada sprint, se aprovado pelo proprietário do produto	Entrega contínua ou a critério da equipe
Funções	Proprietário do produto, mestre scrum, equipe de desenvolvimento	Sem funções existentes. Algumas equipes recorrerem à ajuda de um agile coach.
Principais métricas	Velocidade	Tempo de ciclo
Filosofia de mudança	As equipes devem se esforçar para não fazer alterações na previsão de sprint durante o sprint. Ao fazer isso, o aprendizado em torno da estimativa fica comprometido.	Mudanças podem ocorrer a qualquer momento

Figura 29 - Comparação entre Scrum e Kanban

Mas como será que eles se relacionam? Como eu posso tirar proveito de ambos?

O primeiro item que devemos ressaltar é que ambos são ferramentas de processos. As duas ferramentas objetivam o trabalho mais eficaz.

O segundo item que vale ressaltar é que ambas as ferramentas não são completas e nem perfeitas. Isto fica evidente através da figura 28.

Em terceiro, ambas as ferramentas são adaptativas, mas sendo que o Kanban é menos “engessado” se comparado ao Scrum.

No Scrum, os papéis são definidos e claros (*product owner, scrum master* e time). Já no Kanban não há discriminação dos papéis. Isto não quer dizer que obrigatoriamente deva continuar sem papéis. A existência de papéis é válida, pois deixa mais clara as atribuições de cada participante da iteração.

No Kanban o quadro de atividades é bem similar ao utilizado no Scrum. A diferença é que no Scrum, o quadro é zerado a cada finalização de Sprint. Já no Kanban não há esta preocupação. O único cuidado é a existência de uma coluna denominada *backlog* para tornar visíveis as demandas necessárias para desenvolvimento.

No Scrum, as estimativas geralmente são feitas utilizando pontuação (como foi visto a técnica de *planning poker*). Um item deve estar sempre pontuado para daí seguir para desenvolvimento. Já no Kanban as estimativas são opcionais. O mais importante é manter um baixo número de projetos simultaneamente. Com este controle de WIP, as partes interessadas começam a deixar de esperar tanto.

No Kanban temos um conjunto de dados e gráficos que são mensurados com frequência. Enquanto no Scrum, utilizamos basicamente o gráfico de *burndown*, no Kanban utiliza-se o templo de ciclo, *lead time* e *throughput*.

Para decidir quando utilizar Scrumban (Scrum + Kanban) analise a natureza das suas demandas. Aconselha-se o uso do Scrumban nos seguintes casos:

- Manutenção de projeto,
- Gerenciamento de projetos problemáticos (projetos com histórias de usuários e bugs inesperados),
- Desenvolvimento de novo produto (trabalho que precede o mutirão de desenvolvimento ou a seguir ao mutirão de desenvolvimento) ou
- Gerenciamento contínuo de melhorias.

Lean

Figura 30 – Lean Agile

O *Lean Manufacturing* é reconhecido por ser um método altamente eficaz e que proporciona a entrega de mais valor com menos esforço em cada iteração. Ele é fortemente fundamentado em princípios de **responsabilidade** e **disciplina**. E uma adaptação dele foi realizada para o desenvolvimento de software. Ele é composto por sete princípios:

1) **Eliminação de desperdício**

   Toda etapa deve contribuir para a construção de um produto final com menos custo, mais rapidez e com qualidade. Os desperdícios podem ser categorizados em:

   - Funcionalidades incompletas
   - Códigos incompletos
   - Excesso de processo
   - Criação de documentos
   - Processos complexos
   - Antecipar funcionalidade
   - Troca constante de tarefa
   - Esperas
   - Defeitos

2) **Amplificar o aprendizado (retroalimentação)**
   É a extração das lições aprendidas e das experiências vividas pelo time. Ela gera insumos para uma nova iteração.

3) **Decidir o mais tarde possível**

4) **Entregar o mais rápido possível**

5) Maximizar o ROI (retorno do investimento) através da entrega de software de valor e contínua
6) **Dar poder ao time**

   O ambiente deve existir ou ser criado para que pessoas trabalhem de forma auto-organizada e autodirigida evitando micro gerenciamento

7) **Construir qualidade**
   O desenvolvimento de software deve aplicar técnicas como TDD, refatoração e integração contínua

8) **Otimizar o todo**
   Entender que o software concluído é muito mais que a soma das partes entregues e verificar como ele está alinhado com os objetivos da empresa.

A figura a seguir exibe onde o Lean se encaixa nos modelos ágeis.

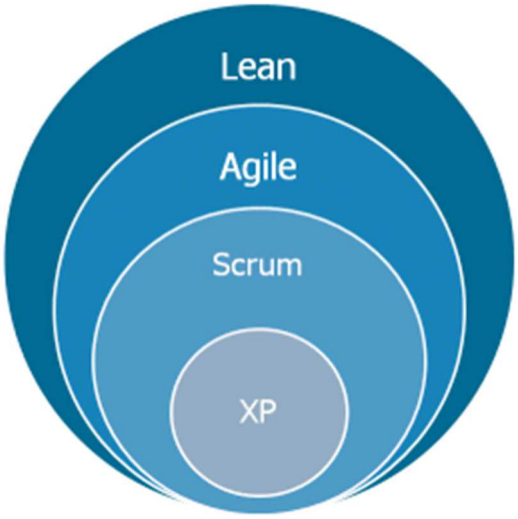


Figura 31 – modelo Lean Agile

Capítulo 4 - Agilidade na codificação – *eXtreme Programming* (XP)

Enquanto Scrum é focado nas práticas de gerenciamento e organização, o *eXtreme Programming* (XP) dá atenção a programação. O XP foi criado por Kent Beck, Ward Cunningham e Ron Jeffries. Ele foi um dos primeiros métodos ágeis. O sucesso do XP vem da busca constante pela satisfação do cliente. Ao invés de entregar tudo o que o cliente possa querer em alguma data no futuro, esse método possibilita a entrega do software que o cliente precisa quando ele precisa. Com XP, os desenvolvedores devem abraçar as mudanças. O XP enfatiza o trabalho em equipe. Gestores, clientes e desenvolvedores são parceiros de igual peso em um time colaborativo.

O XP está dividido em 3 categorias, que são **valores, princípios e prática**.

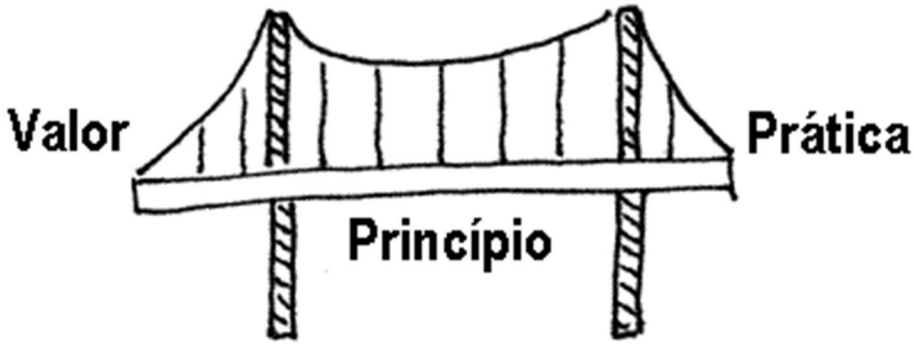


Figura 32 - Divisão do eXtreme Programming

Valores

- **Comunicação**

  O desenvolvimento de software é muito mais que saber programar, é entender as pessoas. Visto isto, a comunicação é essencial para a criação de um produto de qualidade. Esta comunicação deve ser transparente entre desenvolvedores, clientes e usuários.

- **Simplicidade**

  Para manter a simplicidade, faça só o necessário e quando precisar ser feito. Não adicione complexidade antes do tempo. Simplicidade não quer dizer solução de baixa qualidade. Encontrar uma solução simples é mais difícil do que encontrar uma solução.

- **Feedback**

  Este valor é a chave para a criação de software que atende a necessidade do cliente. Ciclos curtos de desenvolvimento e feedback podem garantir o sucesso e o bom andamento das alterações. Em ciclos curtos, os defeitos podem ser corrigidos e já validados rapidamente.

- **Respeito**

As pessoas são os principais elementos para o sucesso do projeto. Elas devem ser respeitadas. O desempenho de cada membro do time está relacionado com a contribuição exercida e as dificuldades individuais são pontos de melhorias da equipe.

- **Coragem**
  É preciso coragem para assumir o comprometimento de mudar, inovar e de aceitar que não sabemos tudo.

Princípios

O XP baseia-se em 14 princípios. Estes princípios transformam os valores em práticas fáceis de implementação no dia a dia do desenvolvimento.

- **Humanidade**

A criação de um software depende do desenvolvedor e suas necessidades individuais devem ser respeitadas e balanceadas com os interesses de negócios e necessidades da equipe

- **Economia**

A equipe deve conhecer as necessidades de negócio e definir prioridade para agregar valor no menor tempo possível.

- **Melhoria**

  Devem ser implementadas constantemente.

- **Benefício mútuo**

Atividades como testes e refatorações beneficiam todos. Investir tempo demasiado em análise com muitas documentações, não gera benefício a todos, já gerará dificuldade na manutenção.

- **Semelhança**

  Boas soluções devem ser aplicadas novamente.

- **Diversidade**

Quanto maior o conjunto de habilidades, opiniões e pontos de vista no time, melhor será para alcançarmos mais perspectivas.

- **Passos pequenos**

A qualidade é mantida quando pequenos passos são desenvolvidos, testados e entregues

- **Reflexão**

A equipe deve refletir sobre suas decisões e aprender com o passado para que a experiência de sucesso repita mais vezes

- **Fluxo**

  Ritmo de trabalho e de entrega de codificação devem ser seguidos.

- **Oportunidade**

  Melhore e esteja atento as oportunidades que surgem.

- **Redundância**

  Testar com frequência para liberação

- **Falha**

  O código nos ensina o tempo todo, por isto experimentamos e não apenas debatemos sobre possíveis abordagens.

- **Qualidade**
  A qualidade não é negociada.

- **Aceitação da responsabilidade**
  Cada membro do time deve aceitar as responsabilidades. As mesmas não devem ser impostas.

Práticas para codificação

As práticas do XP podem ser verificadas no gráfico abaixo:

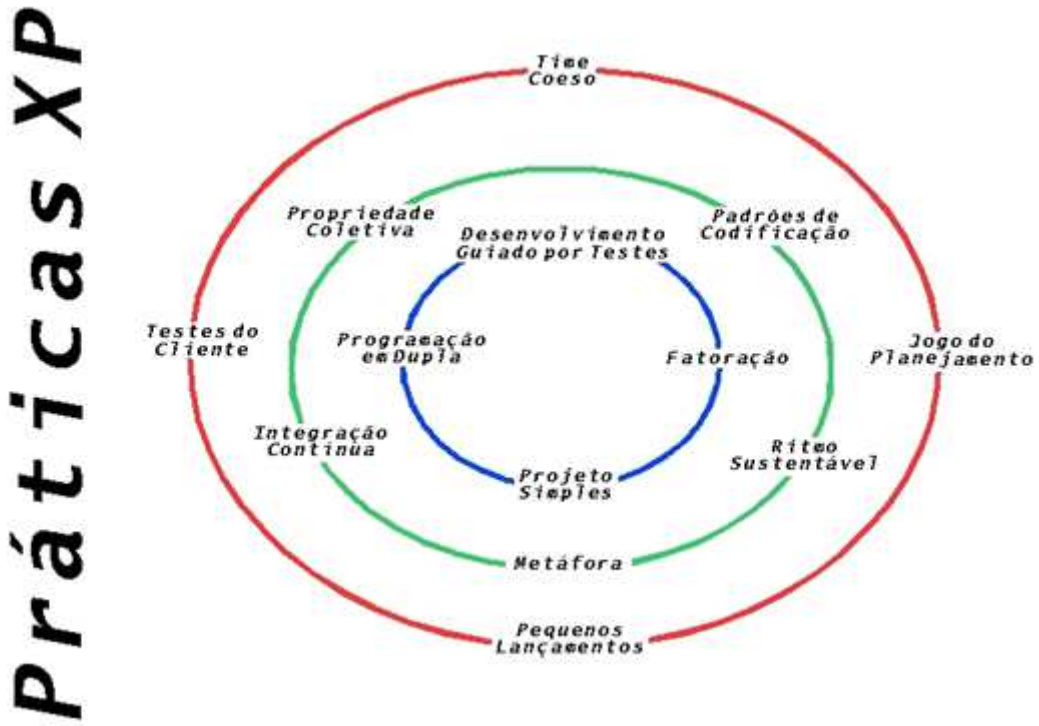


Figura 33 - Práticas XP

Acrescentar novas funcionalidades com frequência requer que a modelagem e arquitetura evoluam a partir de um código organizado e claro. Todos da equipe devem contribuir com os seguintes itens:

- **Padronização do código**
  Definir como o código será escrito. Pode envolver padrões para nomes de variáveis, uso de chaves e colchetes, entre outras opções. Uma sugestão é utilizar o livro *Clean Code* que possuem uma lista de orientações para padronizar o código.

- **Programação pareada**
  Não é só programar. É analisar, projetar, implementar, testar e integrar a funcionalidade. E para isto, o trabalho deve ser feito em dupla. Enquanto um digita o outro revisa. Com isto, a inserção de erros é reduzida e juntamente com a implementação de uma solução inadequada. Como os pares precisam trocar informações, o aprendizado de ambos participantes aumentará.

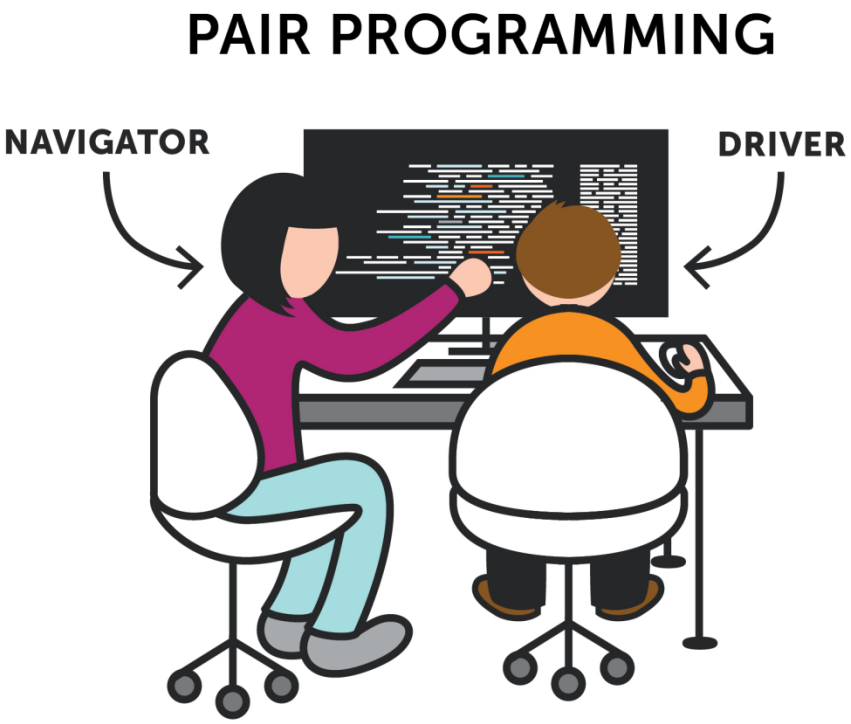


Figura 34 - Pair Programming ou programação em par

Para termos sucesso nesta prática, algumas orientações devem ser seguidas:

1. Utilizar apenas 1 computador e que seja rápido, reduzindo a distração e aperfeiçoando a comunicação dos membros.
2. Utilizar somente o necessário para desenvolver a tarefa. Não abrir outros programas ou sites de redes sociais.
3. Revezar as duplas para que o conhecimento seja disseminado
4. Utilizar formas de pareamento para melhor proveito da prática.

Como formas de pareamento podemos citar:

1. **Baseada em turnos**: através de um tempo estipulado, os papéis se invertem. Quem estava digitando o código (piloto ou *driver*) troca de papel com o co-piloto (ou *navigator*). O tempo não pode ser muito longo. Sugere-se entre 30 segundos a 1 minuto.
2. **Testador – Codificador**: baseia-se na utilização de TDD (*Test Driven Development* ou desenvolvimento baseado em testes). Um desenvolvedor primeiro escreve um teste que falha, então o outro desenvolvedor assume o teclado e deve fazer com que o teste passe escrevendo o mínimo de código possível. Este cria um novo teste que falhe e o teclado retorna ao primeiro desenvolvedor. Este ciclo irá se repetir até a funcionalidade estiver pronta.

Além destas formas, outras podem ser utilizadas. Existem plug-ins para as IDE´s (programas para desenvolvimento de software) que implementam cronômetros e desafios para que o *pair programming* seja aplicado.

Um exemplo é o Pair Hero que é um exemplo de gamificação da prática de pareamento. O objetivo é que colocar em prática a forma de pareamento **Testador – Codificador** e a cada produção de teste ou refatoração, o desenvolvedor ganha pontos e passa o teclado para o seu companheiro, também tentar uma pontuação similar. Após o ciclo de 25 minutos, são comparadas as pontuações. Mas neste game quem fez mais pontos **não é o ganhador**, afinal programação em par não é uma competição é um desenvolvimento colaborativo.

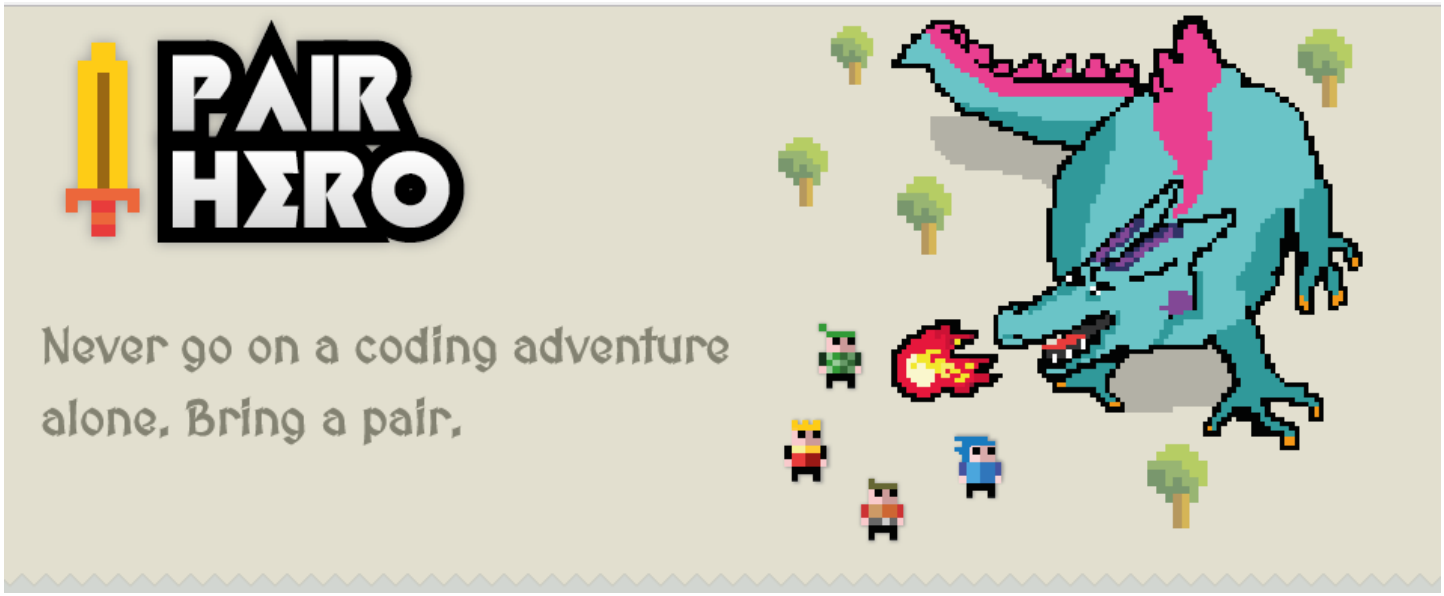


Figura 35 - Pair Hero - um *game* para *pair programming*

Para conhecer mais sobre o Pair Hero acesse http://www.happyprog.com/pairhero/

- **Repositório de código**
  O time deve trabalhar com um repositório com a última versão estável do código fonte.

- **Propriedade coletiva**
  Não podem existir donos de códigos dentro do time. Todos podem modificar tudo.

- **Integração contínua**
  As mudanças em códigos fonte e as refatorações devem ser enviadas com frequência para o repositório para serem validadas e disponibilizadas para os demais membros da equipe

Figura 36 - Ciclo de integração contínua

- **Testes automatizados**

  Cada desenvolvedor deve se tornar o responsável por entregar uma solução testada e pronta para ser usada pelo cliente. Os desenvolvedores devem criar testes unitários para cada parte do sistema. Erros detectados no desenvolvimento causam menos impacto no software do que erros em produção.

- **Refatorações**

  É uma melhoria técnica no código com objetivo de torná-lo mais legível, simples, organizado ou preparado para acomodar novas funcionalidades.

- **Build ágil**

  A construção de uma versão de software não deve ultrapassar 10 minutos e o build deve executar as automatizações de testes e as demais tarefas necessárias para verificar a consistência o código

Para se aprofundar no tema, consulte o livro Programação Extrema (XP) Explicada do autor Kent Beck.

Figura 37 - Capa do Livro Programação extrema (XP)

Referências:

- PRIKLADNICKI, Rafael, WILLI Renato, MILANI, Fabiano - **Métodos ágeis para desenvolvimento de software**. Porto Alegre: Bookman, 2014

- TELES, Vinícius Manhães, **Um Estudo de Caso da Adoção das Práticas e Valores do Extreme Programming**. Orientador: Carlo Emmanoel Tolla de Oliveira. Rio de Janeiro: UFRJ/IM, 2005. Dissertação (Mestrado em Informática). Disponível em http://www.desenvolvimentoagil.com.br/xp/dissertacaoXP.pdf - Acesso em 30/07/2017

- THE STANDISH GROUP INTERNATIONAL, Inc. **Extreme chaos**. The Standish Group International, Inc, 2001. Disponível em: https://courses.cs.ut.ee/MTAT.03.243/2014_spring/uploads/Main/standish.pdf Acesso em 30/07/2017.

- JOHNSON, Jim. "**ROI, It's your job**". Publicado no Third International Conference on Extreme Progrmming, Alghero, Italy, 26-29 de Maio de 2002.

- SCHWABER, Ken, SUTHERLAND, Jeff – **Guia do Scrum** / **Scrum Guide** – 2016 – Disponível em http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Portuguese-Brazilian.pdf - Acesso em 04/08/2017

- JASCONE, Fábio. **Os mitos que sustentam a ignorância ágil**. Publicado no Medium, 03 de julho de 2017. Disponível em https://medium.com/seniortecnologia/os-mitos-que-sustentam-a-ignor%C3%A2ncia-%C3%A1gil-15aec7587d5c . Acesso em 30/07/2017

- AUDY, Jorge. **Guia Rápido de Métodos e Modelos Ágeis**. Disponível em https://www.dropbox.com/s/hp05qzvks54qad6/Super%20Guia%20R%C3%A1pido.pdf?dl=0 - Acesso em 03/08/2017

- **Introdução ao Processo Unificado Aberto** – 01 de abril de 2010. Disponível em http://open2up.blogspot.com.br/2010/04/introducao-ao-processo-unificado-aberto.html – Acesso em 04/08/2017

- **Metodologia Ágil 2.0 – Dos Fundamentos à Ambidestria organizacional** – Objective - Disponível em http://www.objective.com.br/ebook-metodologias-ageis/ - Acesso em 30/07/2017

- KNIBERG, Henrik **- Scrum e XP direto das Trincheiras Como fazemos Scrum** – InfoQ, 2007. ISBN: 978-1-4303-2264-1

- COHN, Mike**, Succeeding with Agile: Software Development Using Scrum,** Porto Alegre: Bookman, 2011

- TAKEUCHI, H. and Nonaka, I. (1986) **The New New Product Development Game**. Harvard Business Review, 64, 137-146. Disponível em https://hbr.org/1986/01/the-new-new-product-development-game - Acesso em 05/08/2017

- **MANIFESTO ÁGIL -** http://www.manifestoagil.com.br/ - Acesso em 30/07/2017

- **DOZE PRINCÍPIOS** - http://www.manifestoagil.com.br/principios.html - Acesso em 30/07/2017

- GRABOSQUE, João Paulo, **Por que duas pizzas são suficientes?,** Disponível em http://www.matera.com/br/2017/03/20/porque-duas-pizzas-sao-suficientes - acesso em 05/08/2017

- JUNIOR, Luiz Fernando Duarte, **Planning Poker: como estimar tempo de desenvolvimento de software**, Disponível em http://www.luiztools.com.br/post/planning-poker-como-estimar-tempo-de-desenvolvimento-de-software - acesso em 05/08/2017

- MARTIN, Robert C. **Clean Code: A Handbook of Agile Software Craftsmanship**. Prentice Hall, 2009

- BOED, Jesper. **Kanban em 10 passos.** InfoQ, 2012

- **BURNDOWN** - https://www.infoq.com/br/news/2010/02/deciphering-burndown-charts - Acesso em 11/08/2017

- KNIBERG, Herik, SKARIN, Mattias. **Kanban e Scrum – obtendo o melhor de ambos**. InfoQ, 2009 . ISBN: 978-0-557-13832-6

- KERTH, Norman L., **Project Retrospectives**. Dorset House,2001.

- DERBY, Esther, LARSEN, Diana, **Agile Retrospectives: Making Good Teams Great**. Pragmatic Bookshelf; 2006.

- **Retro Ágil,** Disponível em https://retroagil.wordpress.com/2015/10/27/porque-retroagil/ - acesso em 17/08/2017

- MARIANO, Pedro, **Programação em Par: Técnicas e dicas**. InfoQ, 2010. Disponível em https://www.infoq.com/br/news/2010/05/tecnicas-programacao-par - acesso em 17/08/2017

Referências das figuras:

1. https://www.agilealliance.org/agile101/subway-map-to-agile-practices/ - acesso em 30/07/2017
2. https://pt.wikipedia.org/wiki/Modelo_em_cascata#/media/File:Modelo_em_cascata.png – acesso em 30/07/2017

3. THE STANDISH GROUP INTERNATIONAL, Inc. **Extreme chaos**. The Standish Group International, Inc, 2001. Disponível em: https://courses.cs.ut.ee/MTAT.03.243/2014_spring/uploads/Main/standish.pdf – acesso em 30/07/2017

4. THE STANDISH GROUP INTERNATIONAL, Inc. **Extreme chaos**. The Standish Group International, Inc, 2001. Disponível em: https://courses.cs.ut.ee/MTAT.03.243/2014_spring/uploads/Main/standish.pdf – acesso em 30/07/2017

5. JOHNSON, Jim. "**ROI, It's your job**". Publicado no Third International Conference on Extreme Progrmming, Alghero, Italy, 26-29 de Maio de 2002.

6. PRIKLADNICKI, Rafael, WILLI Renato, MILANI, Fabiano - **Métodos ágeis para desenvolvimento de software**. Porto Alegre: Bookman, 2014, página xxii.

7. http://sportycious.com/wp-content/uploads/2015/01/A-Beginnerss-Guide-to-the-Basics-of-Rugby-Union-Positions.jpg - acesso em 05/08/2017

8. https://www.scrumalliance.org/scrum/media/ScrumAllianceMedia/PageGraphics/ScrumExplained-4-620.jpg - acesso em 05/08/2017

9. https://www.solutionsiq.com/wp-content/uploads/2016/06/Simple-Scrum-Team-Icons_Artboard-1-300x300.png - acesso em 05/08/2017

10. http://www.blog-gestion-de-projet.com/wp-content/uploads/2016/02/ezuaeiuzieoau.jpg - acesso em 06/08/2017

11. http://a1.mzstatic.com/us/r30/Purple20/v4/92/c7/24/92c72401-2413-a610-c32e-34736de12aa8/sc1024x768.jpeg - acesso em 06/08/2017

12. http://www.gp4us.com.br/wp-content/uploads/2015/11/backlog.png - acesso em 06/08/2017

13. https://s3.amazonaws.com/scrumorg-website-prod/drupal/inline-images/2017-05/ScrumFrameworkTest.png - acesso em 05/08/2017

14. PRIKLADNICKI, Rafael, WILLI Renato, MILANI, Fabiano - **Métodos ágeis para desenvolvimento de software**. Porto Alegre: Bookman, 2014, página 31.

15. https://agilefellow.files.wordpress.com/2016/06/daily-scrum-team.jpg?w=620 – acesso em 06/08/2017

16. http://www.scrum-institute.org/images_scrum/Sprint_Burndown.jpg - acesso em 11/08/2017
17. https://pt.wikipedia.org/wiki/Feature_Driven_Development#/media/File:Fdd.png – acesso em 03/08/2017

18. http://open2up.blogspot.com.br/2010/04/introducao-ao-processo-unificado-aberto.html - acesso em 03/08/2017

19. https://s3.amazonaws.com/static.written.com/kanban-board1488229197.png - acesso em 04/08/2017

20. https://www.culturaagil.com.br/wp-content/uploads/2014/12/kanban-quadro.png - acesso em 17/08/2017

21. http://www.caroli.org/cumulative-flow-diagram/ - acessado em 04/08/2017

22. https://stefanroock.files.wordpress.com/2010/03/leadtimes3.png - acesso em 17/08/2017

23. https://image.slidesharecdn.com/workshopkaizengame-apresentao-170711170145/95/workshop-kanban-22-638.jpg?cb=1499792589 – acesso em 17/08/2017

24. https://cdn-images-1.medium.com/max/800/1*22HzRg7VzZAIvhAArefsUg.png – acesso em 17/08/2017

25. https://confluence.atlassian.com/agile/files/391087320/641597718/3/1424670753619/agile-controlchart-fixedresolutiononly.png - acesso em 17/08/2017

26. Adaptado a partir da figura https://cdn-images-1.medium.com/max/800/1*22HzRg7VzZAIvhAArefsUg.png – acesso em 17/08/2017

27. https://medium.com/@berrondo/diario-kanban-ii-metricas-34ebc3d472e7 - acesso em 04/08/2017

28. https://br.atlassian.com/agile/kanban - acesso em 17/08/2017

29. https://svsg.co/lean-and-agile-partners-in-customer-delight/ - acesso em 04/08/2017

30. http://blog.crisp.se/2008/10/19/henrikkniberg/1224451800000 - acesso em 04/08/2017

31. https://livralivro.com.br/uploads/book/img/875/8536303875.jpg - acesso em 03/08/2017

32. https://devagil.files.wordpress.com/2007/01/ponte2.GIF - acesso em 17/08/2017

33. http://arquivo.devmedia.com.br/artigos/Fabio_Gomes_Rocha/xp/image1.png - acesso em 17/08/2017

34. http://galvanize-wp.s3.amazonaws.com/wp-content/uploads/2016/08/25125440/fullstack-infographic-week-9-pair-programming-01-1.jpg - acesso em 17/08/2017

35. http://www.happyprog.com/pairhero/ - acesso em 17/08/2017

36. https://arcadsoftware.com/wp-content/uploads/2016/05/devops-300x148.png - acesso em 17/08/2017

37. https://livralivro.com.br/uploads/book/img/875/8536303875.jpg - acesso em 04/08/2017