Engenharia de Software

Módulo 04

Prof. Daniel Caixeta

Atualizado em 2.20213

Conteúdo Programático

01 Metodologia Ágil

1.1. Desenvolvimento ágil de *software*.
1.2. Características fundamentais.
1.3. O manifesto ágil.
1.4. Métodos ágeis - 4 valores nos 12 princípios.
1.5. O que é agilidade?

02 Fatores humanos

2.1. Habilidades, competências e atitudes.

03 Ágil *versus* Tradicional

3.1. Comparação entre projetos ágil e tradicional.
3.2. Um resumo.

Metodologia Ágil

Panoramas e novos cenários [...]

1.1. DESENVOLVIMENTO ÁGIL DE *SOFTWARE*

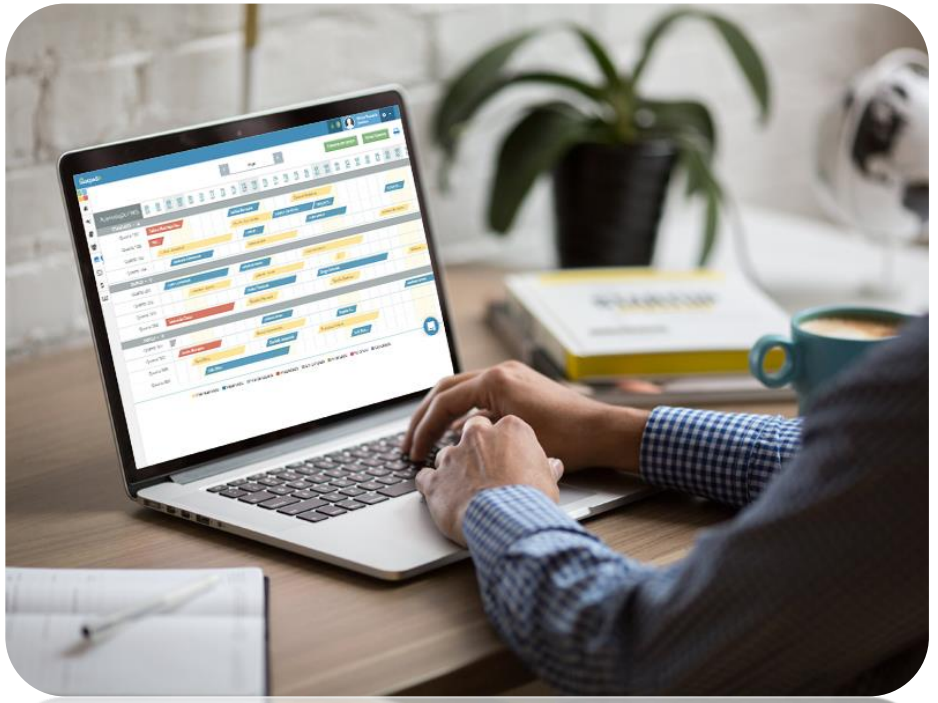
- Segundo Sommerville (2018), nos dias de hoje, as empresas trabalham em ambientes globais, com rápidas mudanças de cenários, respondendo às novas demandas emergenciais e oportunidades em novos mercados.
- *Softwares* fazem parte de quase todas as operações de negócios, assim, são desenvolvidos rapidamente para obterem proveito e responder às pressões competitivas. (*ibidem*).
- Atualmente o desenvolvimento e as entrega são rápidas [...]. Na verdade, muitas empresas estão dispostas a trocar a qualidade e o compromisso com os requisitos do *software* por uma implantação mais rápida de que necessitam. (*ibidem,* grifo meu).
- Esta dinâmica, torna-se em muitos casos impossível de obter um conjunto completo de requisitos para um *software* estável. Devido a estes fatores externos, os requisitos são suscetíveis a mudanças rápidas e imprevisíveis.

- Processos que planejam especificar completamente os requisitos e, em seguida, projetar, construir e testar não estão adaptados ao desenvolvimento ágil.
- Já, nos processos convencionais/tradicionais, *e.g.*, em cascata ou baseado em especificações, costumam serem mais demorados, e o *software* final poderá ser entregue ao cliente bem depois do prazo acordado.
- Por exemplo:

  Para alguns tipos de *software*, como sistemas críticos de controle de segurança, é essencial uma análise completa, por isso que uma abordagem dirigida a planos é a melhor opção. No entanto, em um ambiente de negócio que se caracteriza por mudanças rápidas, isso talvez não seja possível.

cont. [...]

Sala de missões da NASA.



Software para reservas em hotéis.

1.2. CARACTERÍSTICAS FUNDAMENTAIS

- Os processos ágeis são concebidos para produzir, rapidamente, *softwares* úteis.
- Um *software* não é desenvolvido como uma única unidade, mas sim como uma série de incrementos, onde cada incremento inclui uma nova funcionalidade do sistema.
- Embora existam muitas abordagens, todas compartilham algumas características fundamentais:
  - Os processos de especificação, projeto e implementação são intercalados, ou seja, não há especificação detalhada do sistema.
  - A documentação do projeto é minimizada ou gerada automaticamente pelo ambiente de programação usado para implementar o sistema, já o documento de requisitos do usuário apenas define as características mais importantes do sistema.
  - O sistema é desenvolvido em uma série de versões.

- Os usuários finais e outros *stakeholders* do sistema são envolvidos na especificação e avaliação de cada versão, e podem propor alterações ao *software* e novos requisitos que devem ser implementados em uma versão posterior do sistema. [...]
- As interfaces de usuário do sistema são geralmente desenvolvidas de forma interativa permitindo a rápida criação do projeto por meio de desenho e posicionamento de ícones na interface.

- Os métodos ágeis são métodos de desenvolvimento incremental em que os incrementos são pequenos e, normalmente, as novas versões do sistema são criadas e disponibilizadas aos clientes a cada duas ou três semanas. (SOMMERVILLE, 2018).
- Envolvem os clientes no processo de desenvolvimento para obter *feedback* rápido sobre a evolução dos requisitos. (*ibidem*).

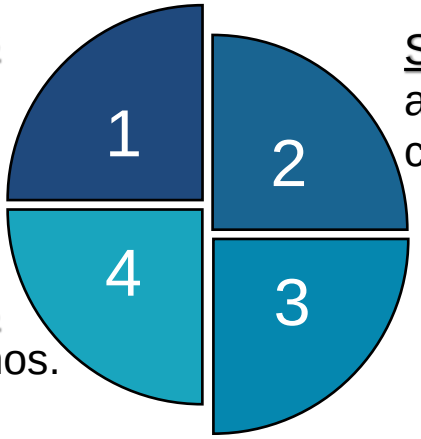
1.3. O MANIFESTO ÁGIL

- Em 2001, Kent Beck e outros dezesseis renomados desenvolvedores, autores e consultores da área de *software* (batizados de *Agile Alliance*/Aliança dos Ágeis) assinaram o "Manifesto para o Desenvolvimento Ágil de Software" ("*Manifesto for Agile Software Development*")[1], que se inicia da seguinte maneira (PRESSMAN, 2011):

"Desenvolvendo e ajudando outros a desenvolver *software*, estamos desvendando formas melhores de desenvolvimento. Por meio deste trabalho passamos a valorizar":

<u>Indivíduos e interações</u>
acima de processos e ferramentas.

<u>Software funcionando</u>
acima de documentação compreensível.



<u>Respostas a mudanças</u>
acima de seguir os planos.

<u>Colaboração dos clientes</u>
acima de negociação contratual.

Embora haja valor nos textos abaixo, valorizaremos os textos de cima ainda mais (acima, negrito).

1. https://agilemanifesto.org/iso/ptbr/manifesto.html

1.4. MÉTODOS ÁGEIS - 4 VALORES NOS 12 PRINCÍPIOS.

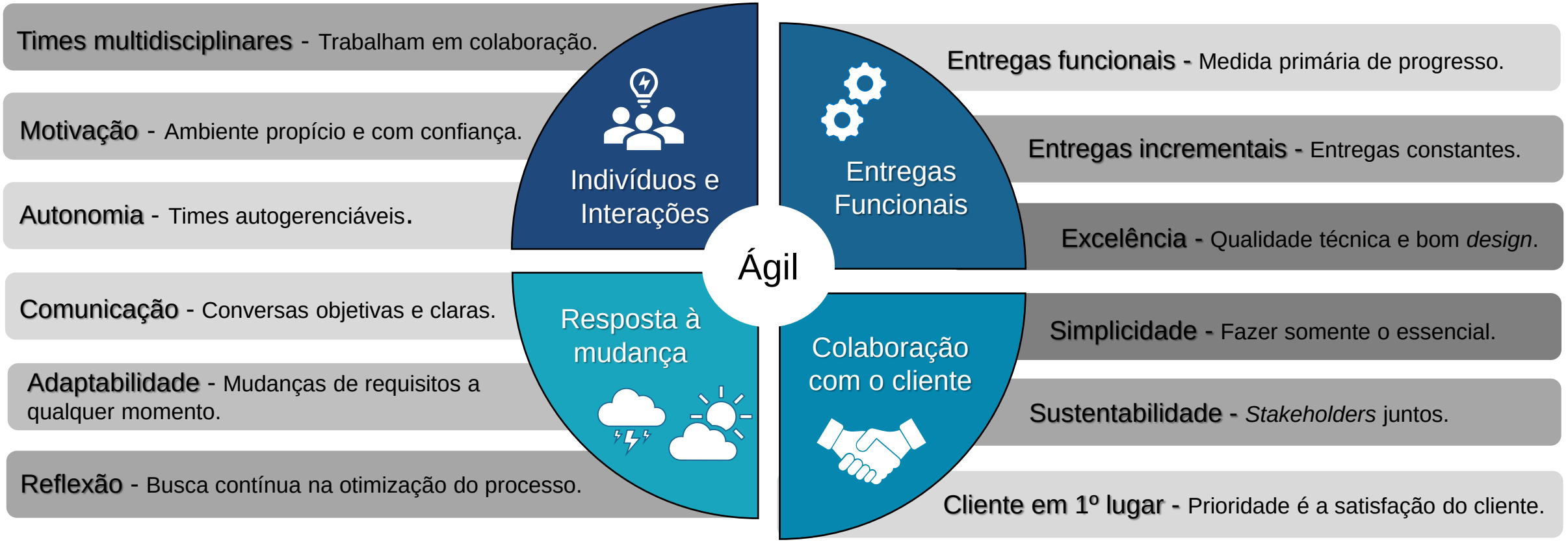


Figura 1. Resumo dos 12 princípios nos 4 valores dos métodos ágeis. (InovaLab 2018 *adaptado* CAIXETA, 2020).

1.5. O QUE É AGILIDADE?

- No contexto da engenharia de software, o que é agilidade? Jacobson (2002, *apud* PRESMANN, 2011) apresenta uma discussão útil:

> Atualmente, *agilidade* tornou-se a palavra da moda quando se descreve um moderno processo de software. Todo mundo é ágil. Uma equipe ágil é aquela rápida e capaz de responder apropriadamente a mudanças. Mudanças têm muito a ver com desenvolvimento de software. Mudanças no software que está sendo criado, mudanças nos membros da equipe, mudanças devido a novas tecnologias [...]. Uma equipe ágil reconhece que o software é desenvolvido por indivíduos trabalhando em equipes e que as habilidades dessas pessoas, suas capacidades em colaborar estão no cerne do sucesso do projeto.

- Porém, agilidade consiste em algo mais que uma resposta à mudança. Abrange uma filosofia como aquela proposta no manifesto.
- Enfatiza rápidas entregas do *software* operacional e diminui a importância dos artefatos intermediários.
- Assume o cliente como parte da equipe de desenvolvimento.

Fatores humanos

Competências, habilidades e atitudes.

2. OS FATORES HUMANOS: COMPETÊNCIA, HABILIDADE E ATITUDE

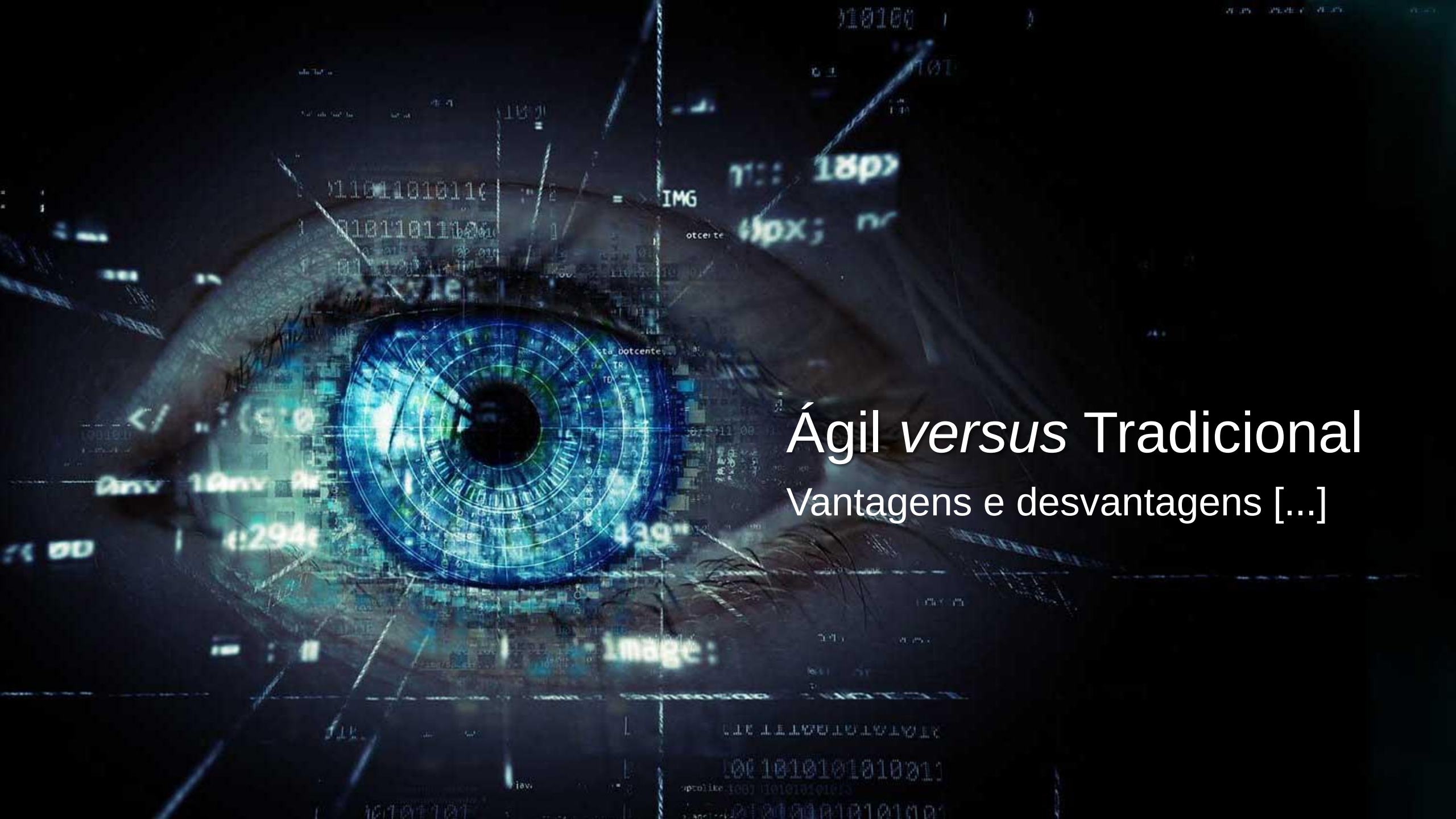
- Um dos principais fatores no desenvolvimento ágil é a ênfase no potencial humano. Como afirmam Cockburn & Highsmith (2001, *apud* PRESSMAN, 2011, p.86), “O desenvolvimento ágil foca talentos e habilidades de indivíduos, moldando o processo de acordo com as pessoas e as equipes específicas”.
- O ponto chave nessa afirmação é que o processo se molda às necessidades das pessoas e equipes, e não o caminho inverso.

> Muito da agilidade dos métodos deriva do fato de terem suas bases no conhecimento tácito incorporado pela e na equipe, em vez de registrar por escrito tal conhecimento em planejamentos. (Barry Boehm).

- Para Pressman (p.86) as equipes/times devem ser auto-organizada e reguladoras no controle de todas as atividades executadas. Ela estabelece seus próprios compromissos e define planos para cumpri-los.

- No perfil dos membros de uma equipe ágil devem conter:

Figura 2. Habilidades/*skills* de um membro de equipe ágil. (CAIXETA, 2020).



Ágil *versus* Tradicional

Vantagens e desvantagens [...]

3.1. PROJETOS: ÁGIL x TRADICIONAL

Figura 3. Comparação entre projetos ágeis *versus* tradicional. (InovaLab 2018, *adaptado* CAIXETA, 2020).

3.2. UM RESUMO

- Os modelos ágeis seguem uma filosofia diferente da filosofia dos modelos preditivo (tradicionais). Em vez de apresentar uma “receita de bolo”, com fases ou tarefas a serem executadas, eles focam valores humanos e sociais. (WAZLAWICK, 2013).
- Apesar de os métodos ágeis serem usualmente mais leves, é errado entendê-los como modelos de processos menos complexos ou simplistas. Não se trata apenas de simplicidade, mas de focar mais nos resultados do que no processo. (*ibidem*, grifo meu).
- Isso não significa que os modelos ágeis não valorizem processos, ferramentas, documentação, contratos e planos. Quer dizer apenas que esses elementos terão mais sentido e mais valor depois que indivíduos, interações, *software* funcionando, colaboração do cliente e resposta às mudanças também forem considerados importantes. (*ibidem*).
- Veremos nas próximas seções exemplos de modelos ágeis: FDD, DSDM, *Scrum*, XP, ASD e LSD.

Os modelos Ágeis [...]

Alguns exemplos [...].

4.1. FDD - *Feature-Driven Development*.

- FDD[4] (*Feature-Driven Development*/Desenvolvimento Dirigido por Funcionalidade) é um método ágil que enfatiza o uso de orientação a objetos. (WAZLAWICK, 2013).
- Possui apenas duas grandes fases:
    1. Concepção e planejamento: Implica em pensar um pouco (em geral de 1 a 2 semanas) antes de começar a construir.
    2. Construção: Desenvolvimento iterativo com ciclos de 1 a 2 semanas.
- Como outras abordagens ágeis, o FDD adota uma filosofia que:
    1. Enfatiza a colaboração entre pessoas da equipe;
    2. Gerencia problemas e complexidade de projetos utilizando a decomposição baseada em funcionalidades, seguida pela integração dos incrementos de *software*, e;
    3. Comunicação de detalhes técnicos usando meios verbais, gráficos e de texto.

2. Disponível em: <www.featuredrivendevelopment.com/>. Acessado em: 16 ago. 2023.

cont. [...]

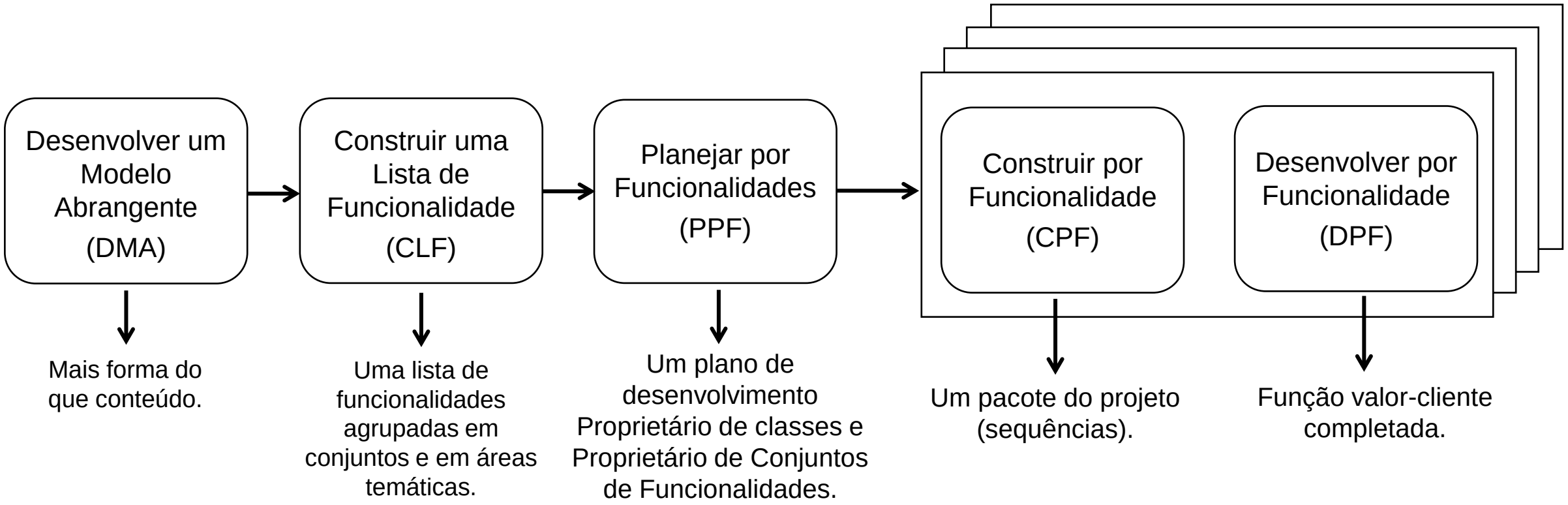


Figura 4. Modelo FDD segundo Pressman (2011, *adaptado* CAIXETA, 2020).

cont. [...]

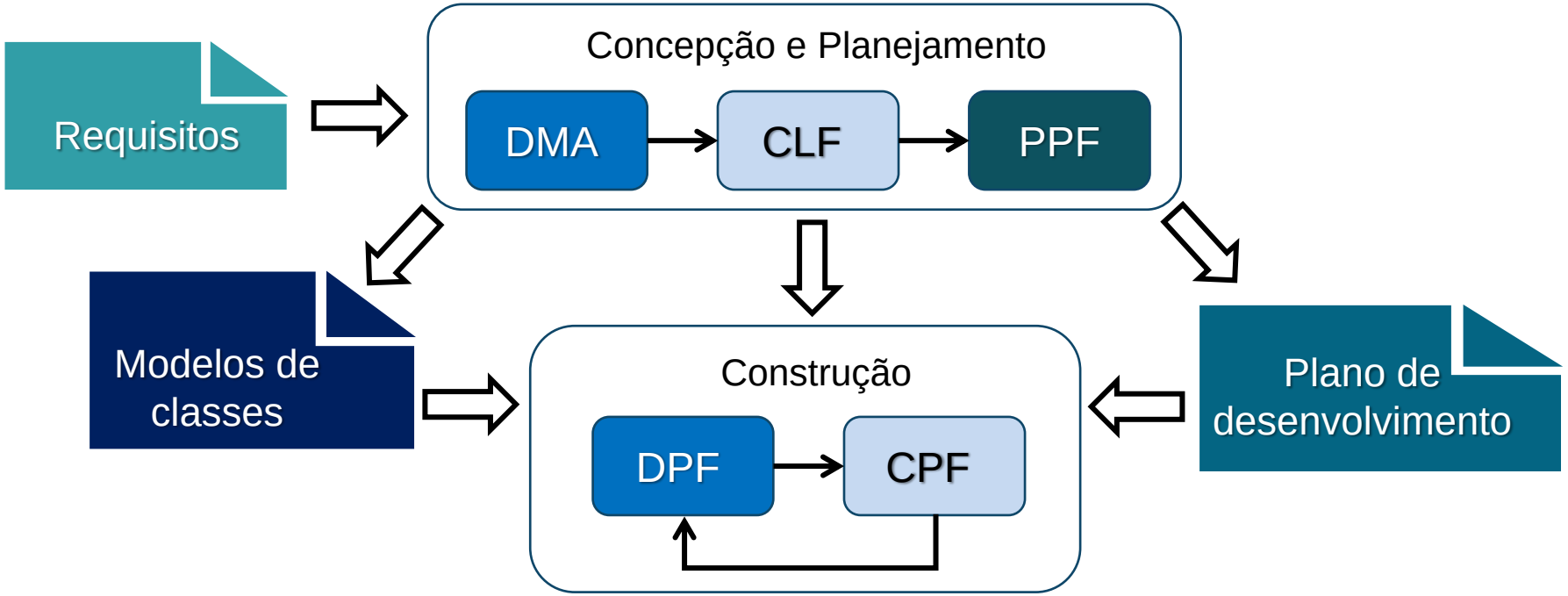


Figura 5. Modelo FDD segundo Wazlawick (2013 *adaptado* CAIXETA, 2020).

- DMA – Desenvolver Modelo Abrangente.
- CLF – Construir Lista de Funcionalidades.
- PPF – Planejar por Funcionalidade.
- DPF – Detalhar por Funcionalidade.
- CPF – Construir por Funcionalidade.

4.2. DSDM - *Dynamic System Development Method.*

- *Dynamic Systems Development Method* (DSDM)[5] também é um modelo ágil baseado em desenvolvimento iterativo e incremental, com participação ativa do usuário. (WAZLAWICK, 2013).
- O DSDM possui três fases:
    a. Pré-projeto: Nesta fase, o projeto é identificado e negociado, seu orçamento é definido e um contrato é assinado.
    b. Ciclo de vida: O ciclo de vida se inicia com uma fase de análise de viabilidade e de negócio. Depois entra em ciclos iterativos de desenvolvimento.
    c. Pós-projeto: Equivale ao período normalmente considerado como operação ou manutenção. Nessa fase, a evolução do *software* é vista como uma continuação do processo de desenvolvimento, podendo, inclusive, retomar fases anteriores, se necessário.

3. Disponível em: <https://flowup.me/blog/dsdm/>. Acessado em: 16 ago. 2023.

- A fase de ciclo de vida, por sua vez, subdivide-se nos seguintes estágios:
  a. Análise de viabilidade.
  b. Análise de negócio.

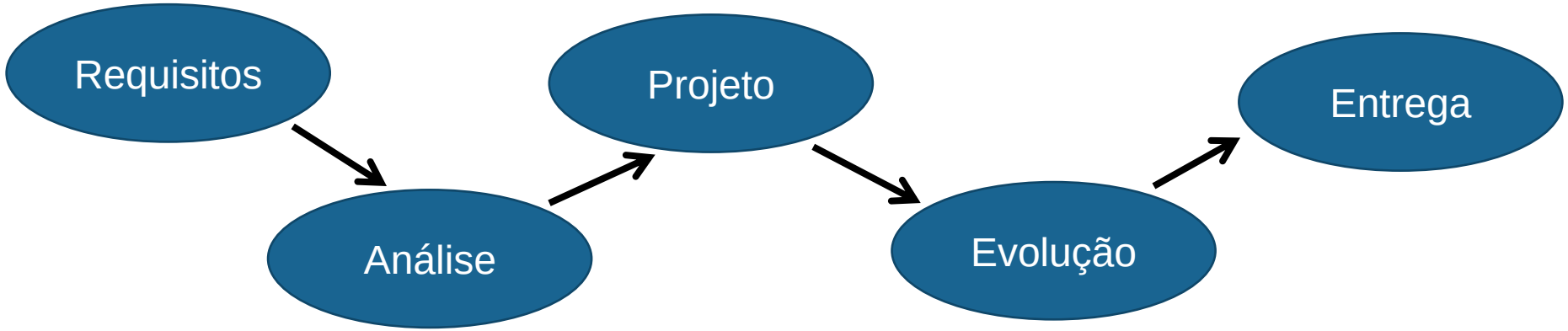
  Normalmente a viabilidade e negócio são analisados em conjunto.
  c. Iteração do modelo funcional.
  d. Iteração de elaboração e construção.
  e. Implantação.
- O DSDM fundamenta-se no Princípio de Pareto[6] (20/80). Assim, o DSDM procura iniciar pelo estudo e implementação dos 20% dos requisitos que serão mais determinantes para o sistema como um todo. Essa prática é compatível com a prática de abordar inicialmente requisitos mais complexos ou de mais alto risco, presentes em outros modelos, como UP. (WAZLAWICK, 2013).

4. O princípio foi originalmente formulado pelo economista italiano Alfredo Pareto (século XIX), que observou que, em muitas situações, 80% das consequências são devidas a 20% das causas. Em engenharia de requisitos, pode-se verificar que, em geral, 80% do sistema advém de 20% dos requisitos. (Nota de rodapé, WAZLAWICK, 2013).

- O DSDM não é recomendável para projetos nos quais a segurança é um fator crítico, pois a necessidade de testes exaustivos desse tipo de sistema entra em conflito com os objetivos de custo e prazo do DSDM. (WAZLAWICK, 2013).
- É bem mais fortemente baseado em documentação do que o *Scrum* ou o XP, o que até o deixa mais parecido com o Processo Unificado do que com os métodos ágeis. (*ibidem*).
- Já uma das características do DSDM que o diferenciam do Processo Unificado, é o fato de que ele não preconiza o uso de nenhuma técnica específica. Ele é, na verdade, um *framework* de processo, no qual os participantes poderão desenvolver suas atividades utilizando suas técnicas preferidas. (*ibidem*).

4.3. SCRUM - GERENCIAMENTO ÁGIL DE PROJETOS

- *Scrum* é um modelo ágil para a gestão de projetos de *software*. (WAZLAWICK, 2013).
- No *Scrum* um dos conceitos mais importantes é o *sprint*, que consiste em um ciclo de desenvolvimento que, em geral, vai de duas semanas a um mês. (*ibidem*).
- Esta concepção surgiu na indústria automobilística (Takeuchi & Nonaka, 1986), e foi adaptado a várias outras áreas, inclusive na de produção de *software*. (*ibidem*).
- De acordo com Pressman (2011), os princípios do *Scrum* são consistentes com o manifesto ágil e são usados para orientar as atividades de desenvolvimento dentro de um processo que incorpora as seguintes atividades estruturais:



cont. [...]

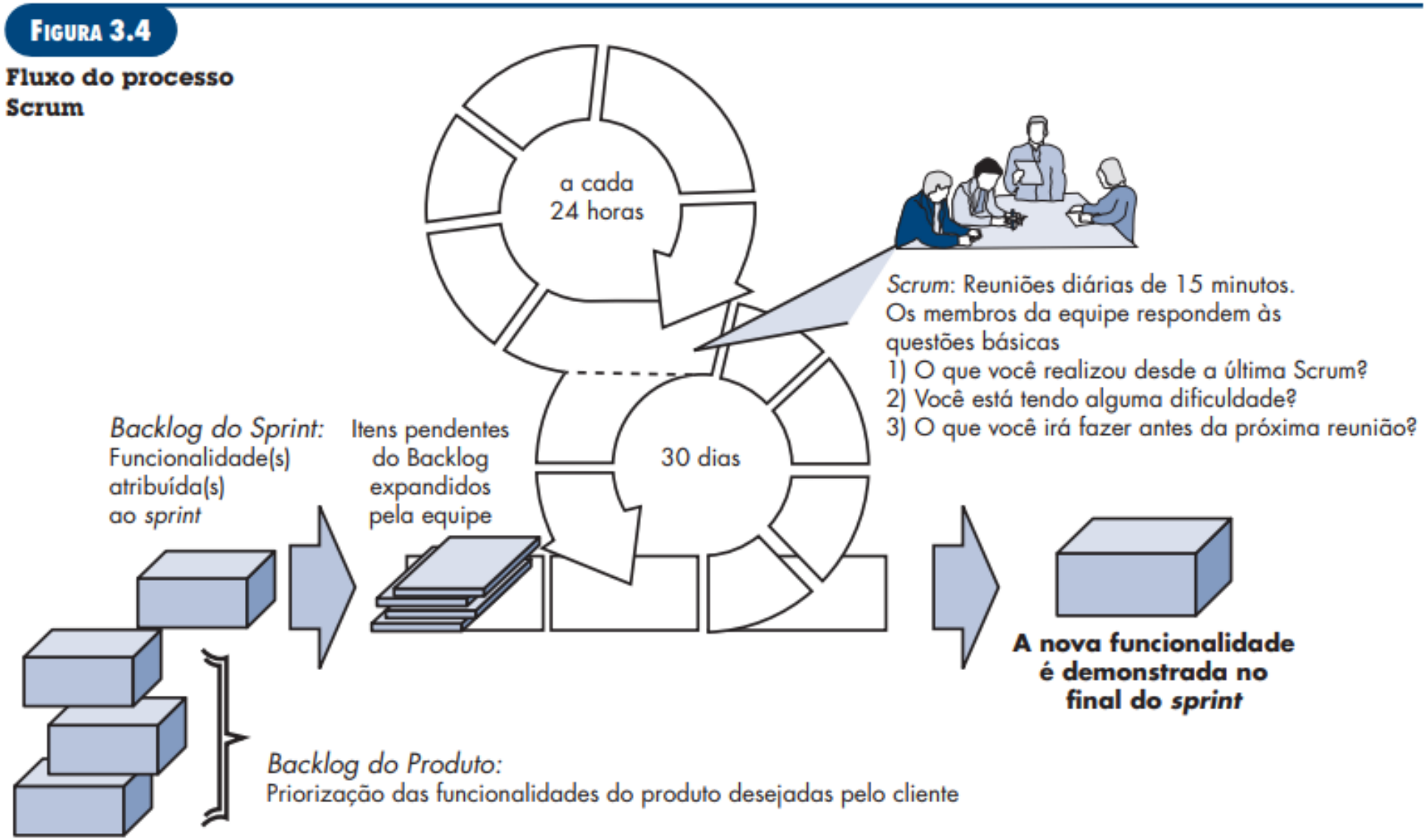


Figura 6. Fluxo geral do processo *Scrum*. (PRESSMAN, 2011).

4.3.1. Os perfis

- Há três perfis importantes no modelo *Scrum*:

1. O *Scrum master*, que não é um gerente no sentido dos modelos prescritivos. Não é um líder, já que as equipes são auto-organizadas, mas um facilitador (pessoa que conhece bem o modelo) e um solucionador de conflitos.

2. O *Product owner* é a pessoa responsável pelo projeto em si. Tem, entre outras atribuições, a de indicar quais são os requisitos mais importantes a serem tratados em cada *sprint*. O *Product owner* é o responsável pelo ROI (*Return On Investment*) e também por conhecer e avaliar as necessidades do cliente.

3. *Scrum team* é a equipe de desenvolvimento. Essa equipe não é necessariamente dividida em papéis como analista, *designer* e programador, mas todos interagem para desenvolver o produto em conjunto. Em geral são recomendadas equipes de 6 a 10 pessoas.

- No caso de projetos muito grandes, aplica-se o conceito de *Scrum of Scrums*, em que vários *Scrum teams* trabalham em paralelo e cada um contribui com uma pessoa para a formação do *Scrum of Scrums.* Há sincronização entre as equipes.

4.3.2. *Product backlog* (Registro pendentes de trabalhos)

- De acordo com Pressman (2011):

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

- Um dos princípios do manifesto ágil é usado aqui: adaptação em vez de planejamento. Por isso que o *product backlog* não precisa ser completo no início do projeto. Pode-se iniciar apenas com as funcionalidades mais evidentes, aplicando o princípio de Pareto (20/80), para depois, à medida que o projeto avançar, tratar novas funcionalidades que forem sendo descobertas. [...]. Deve-se tentar obter com o cliente o maior número possível de informações sobre suas necessidades. (WAZLAWICK, 2013).

cont. [...]

Exemplo de um *Product backlog*

ERP Veterinário e Pet shop					
ID	Nome	Imp. (1 a 10)	PH[7]	Como demonstrar	Notas
1	Acessar	10	4 (2p x 2d)	Acessar, abrir a página de *login*, digitar id. usuário e senha. Ir para a página inicial do sistema. Diferenças entre *user* comum e adm.	Precisa dos diagramas UML de caso de uso, classe e sequência. No momento não é preciso se preocupar com criptografia.
2	Cadastrar	7	8 (2p x 4d)	Abrir seção cadastrar cliente. Permissão adm. Digitar informações necessárias no perfil do cliente. Salva informações no BD.	Precisa do BD com CRUD implementado. Diagramas de caso de uso, classe e sequência. Não preocupar-se com criptografia.

5. *Pontos de histórias* (PH) é a estimativa de esforço muito utilizada nos métodos ágeis como *Scrum* e XP. Não é uma medida de complexidade funcional, (*e.g.*, pontos de função ou pontos de caso de uso), mas uma medida de esforço relativa à equipe de desenvolvimento. Segundo Kniberg (2007 *apud* Wazlawick, 2013), a estimativa deve ser feita pela equipe. Inicialmente, pergunta-se quanto tempo x pessoas levariam para gerar uma versão executável funcional. Se a resposta for, por exemplo, “3 pessoas levariam 4 dias”, então atribua à história 3 x 4 = 12 PH. Assim, um ponto de história pode ser definido como o esforço de desenvolvimento de uma pessoa durante um dia ideal de trabalho, com dedicação de 6 a 8 horas a um projeto, sem interrupções nem atividades paralelas.

4.3.3. *Sprint*

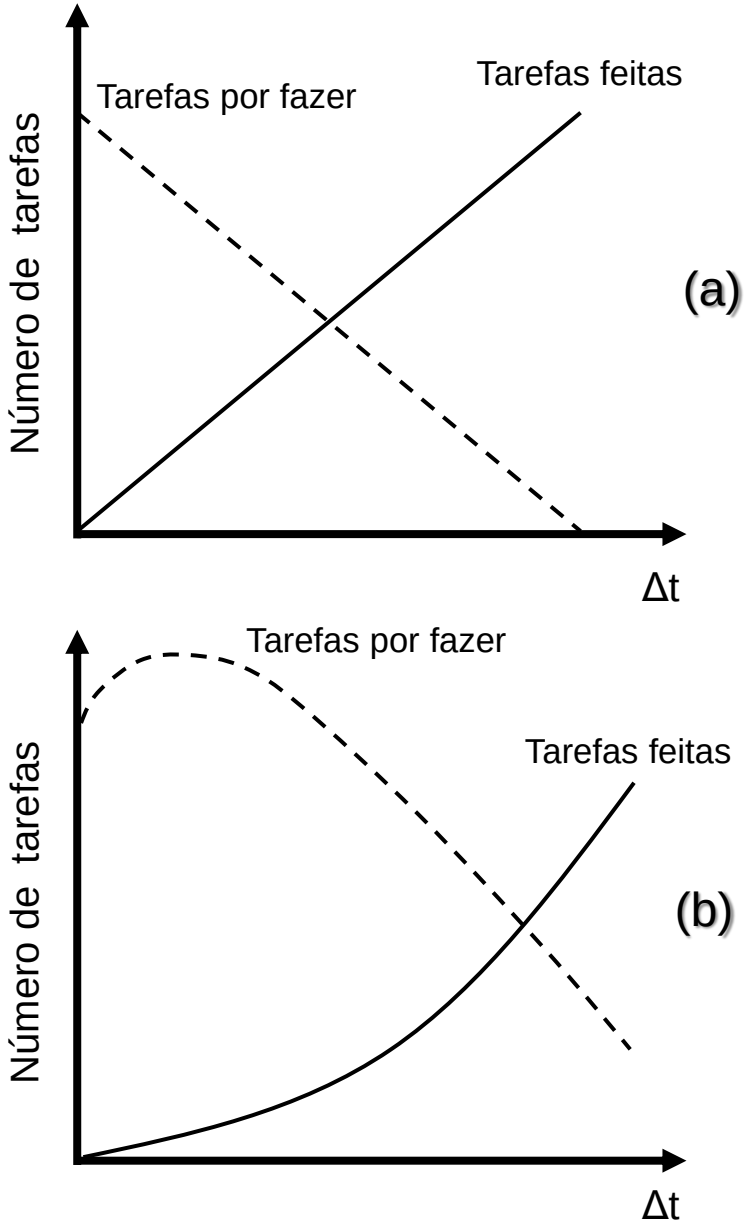
- O *sprint* é um ciclo de desenvolvimento com poucas semanas de duração.
- No início é realizada um *sprint planning meeting*[8], na qual a equipe prioriza os elementos do *product backlog* a serem implementados e transfere esses elementos para o *sprint backlog*, ou seja, a lista de funcionalidades a serem implementadas no ciclo que se inicia. (WAZLAWICK, 2013).
- Um detalhe importante: Cada equipe se compromete em desenvolver as funcionalidades, e o *product owner* de não trazer novas funcionalidades durante o mesmo *sprint*. Se novas funcionalidades forem descobertas, serão abordadas em *sprints* posteriores.
- Portanto, podemos dizer que os dois *backlogs* têm naturezas distintas:
    1. O *product backlog* apresenta requisitos de alto nível - voltados às necessidades diretas do cliente.
    2. Já o *sprint backlog* apresenta uma visão desses requisitos de forma mais voltada à maneira como a equipe vai desenvolvê-los.

6. Média de 15 minutos.

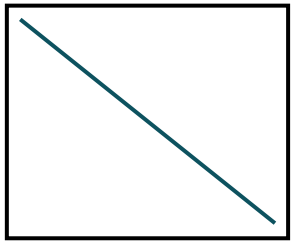
- É função do *product owner* manter o *sprint backlog* sempre atualizado, indicando que tarefas já foram concluídas e aquelas que ainda estão por concluir, preferencialmente mostradas em um gráfico atualizado diariamente e à vista de todos. (WAZLAWICK, 2013). (Exemplos de gráficos To do, Todoist, Kanban, etc.).
- A cada dia também pode-se avaliar o andamento das atividades, contando a quantidade de atividades por fazer e a quantidade de atividades terminadas, o que vai produzir o diagrama *sprint burndown*.
- Em relação aos gráficos ao lado temos:

  (a) *Sprint* com *burndown* ideal.

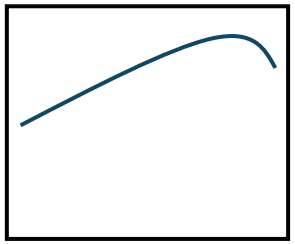
  (b) *Sprint burndown* em que as tarefas adicionais são incluídas após o início do *sprint*.



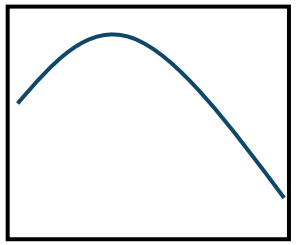
- Kane Mar (2006 *apud* WAZLAWICK, 2013) analisa as linhas de tarefas por fazer, identificando sete tipos de comportamentos de equipes conforme seus *sprint burndowns*. São eles:

1) 

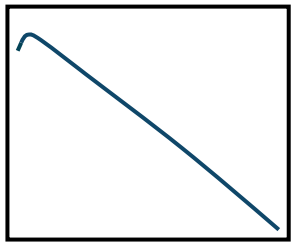
Fakey-fakey: Caracteriza-se por uma linha reta e regular que indica que provavelmente a equipe não está sendo muito honesta, porque o mundo real é bem complexo e dificilmente projetos se comportam com tanta regularidade.

2) 

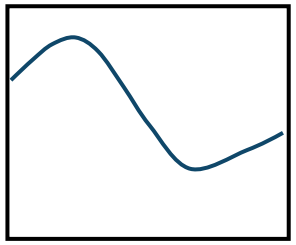
Late-learner: Indica um acúmulo de tarefas até perto do final do *sprint*. É típico de equipes iniciantes que ainda estão tentando aprender o funcionamento do *Scrum*.

3) 

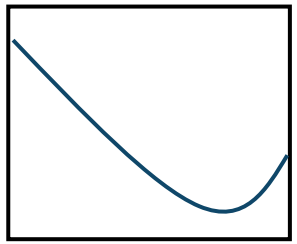
Middle-learner: Indica que a equipe pode estar amadurecendo e começando mais cedo as atividades de descoberta e, especialmente, os testes necessários.

4) 

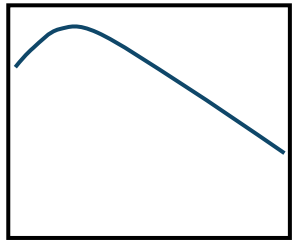
Early-learner: Indica uma equipe que procura, logo no início do *sprint*, descobre todas as necessidades e depois desenvolve-as gradualmente até o final do ciclo.

5) 

Plateau: Indica uma equipe que tenta balancear os aprendizados precoce e tardio, o que acaba levando o ritmo de desenvolvimento a um platô. Inicialmente, fazem bom progresso, mas não conseguem manter o ritmo até o final do *sprint*.

6) 

Never-never: Indica uma equipe que acaba tendo surpresas desagradáveis no final de um *sprint*.

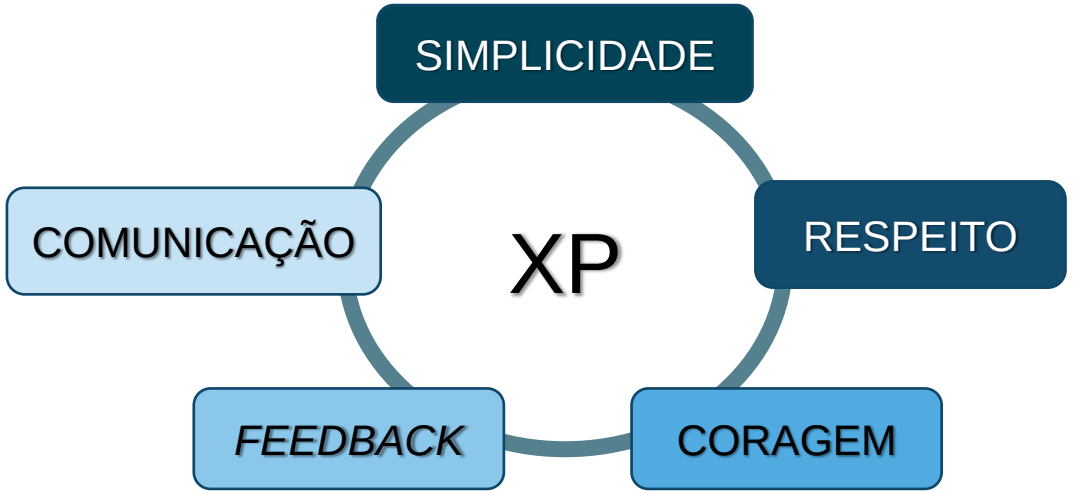
7) 

Scope increase: Indica uma equipe que percebe um súbito aumento na carga de trabalho por fazer. Usualmente, a solução nesses casos é tentar renegociar o escopo da *sprint* com o *product owner*, mas não se descarta também uma finalização da *sprint* para que seja feito um replanejamento do *product backlog*.

- Ao final de cada *sprint*, a equipe deve realizar um *sprint review meeting* (ou *sprint demo*) para verificar o que foi feito e, então, partir para uma nova *sprint*. (WAZLAWICK, 2013).
- O *sprint review meeting* é a demonstração e a avaliação do produto da *sprint*. (*ibidem*).
- Outra reunião que pode ser feita ao final de uma *sprint* é a *sprint retrospective*, cujo objetivo é avaliar a equipe e os processos (impedimentos, problemas, dificuldades, ideias novas, etc.). (*ibidem*).

4.4. XP (*eXtreme Programming*)

- Programação Extrema, ou XP (*eXtreme Programming*) é um modelo ágil inicialmente adequado a equipes pequenas e média baseado em uma série de valores, princípios e regras. (WAZLAWICK, 2013). Surgiu nos Estados Unidos no final da década de 1990.
- Entre os principais valores do XP podemos citar:



- A partir desses valores, uma série de princípios básicos foram definidas:

Princípios XP
1. *Feedback* rápido.
2. Presumir simplicidade.
3. Mudanças incrementais.
4. Abraçar mudanças.
5. Trabalho de alta qualidade.

- O XP preconiza mudanças incrementais e *feedback* rápido, além de considerar a mudança algo positivo, que deve ser entendido como parte do processo.
- Valoriza o aspecto da qualidade.
- A esses princípios pode-se adicionar ainda a priorização de funcionalidades mais importantes, de forma que, se o trabalho não puder ser todo concluído, pelo menos as partes mais importantes terão sido.

4.4.1. As práticas do XP

- Para aplicar o XP, é necessário seguir uma série de práticas que dizem respeito ao relacionamento com o cliente, a gerência do projeto, a programação e os testes. (WAZLAWICK, 2013):

  a. Jogo de planejamento (*Planning game*): Reuniões semanais com o cliente para priorizar as funcionalidades a serem desenvolvidas. O cliente identifica as principais necessidades e a equipe desenvolve o que pode ser implementado no ciclo semanal, com entregas até o final da semana. Esse tipo de modelo de relacionamento com o cliente é adaptativo.

  b. Metáfora (*Metaphor*): É importante conhecer a linguagem do cliente e seus significados. A equipe deve aprender a se comunicar de forma que ele compreenda.

  c. Equipe coesa (*Whole team*): O cliente faz parte da equipe de desenvolvimento e as eventuais barreiras de comunicação devem ser eliminadas.

  d. Reuniões em pé (*Stand-up meeting*): Como no caso do *Scrum*, reuniões em pé tendem a ser mais objetivas e efetivas.

e. *Design* simples (*Simple design*): Implica atender a funcionalidade solicitada pelo cliente sem sofisticar demais. Deve-se fazer aquilo que o cliente precisa, não o que o desenvolvedor gostaria que ele precisasse. Por vezes, *design* simples pode ser confundido com *design* fácil.

f. Versões pequenas (*Small releases*): A liberação de pequenas versões do sistema pode ajudar o cliente a testar as funcionalidades de forma contínua. O XP leva esse princípio ao extremo, sugerindo versões ainda menores do que as de outros processos incrementais, como UP e *Scrum*.

g. Ritmo sustentável (*Sustainable pace*): Trabalhar com qualidade não mais que 8 horas diárias. Horas extras só são recomendadas quando efetivamente trouxerem aumento de produtividade, mas não podem ser rotina.

h. Posse coletiva (*Collective ownership*): O código não tem dono e não é necessário pedir permissão a ninguém para modificá-lo.

i. Padrões de codificação (*Coding standards*): Deve estabelecer e seguir padrões de codificação, de forma que o código pareça ter sido todo desenvolvido pela mesma pessoa, mesmo que tenha sido feito por dezenas delas.

j. Programação em pares (*Pair programming*): O desenvolvimento do código é sempre feita por duas pessoas em cada computador, em geral um programador mais experiente e um aprendiz. [...]. Com isso, o código gerado será sempre sido verificado por pelo menos duas pessoas, reduzindo drasticamente a possibilidade de erros.

k. Testes de aceitação (*Customer tests*): São testes planejados e conduzidos pela equipe em conjunto com o cliente para verificar se os requisitos foram atendidos.

l. Desenvolvimento orientado a testes (*Test Driven Development* - TDD): Antes de programar uma unidade, devem-se definir e implementar os testes pelos quais ela deverá passar.

m. Refatoração (*Refactoring*): Trata-se de um processo de melhoramento do código e não implementação de novas funcionalidade. Essas revisões sistemáticas permitem a diminuição de erros. (CAIXETA, 2023).

n. Integração contínua (*Continuous integration*): Nunca se deve esperar até o final do ciclo para integrar uma nova funcionalidade. Assim que estiver viável, ela deverá ser integrada ao sistema para evitar surpresas.

4.4.2. O processo XP

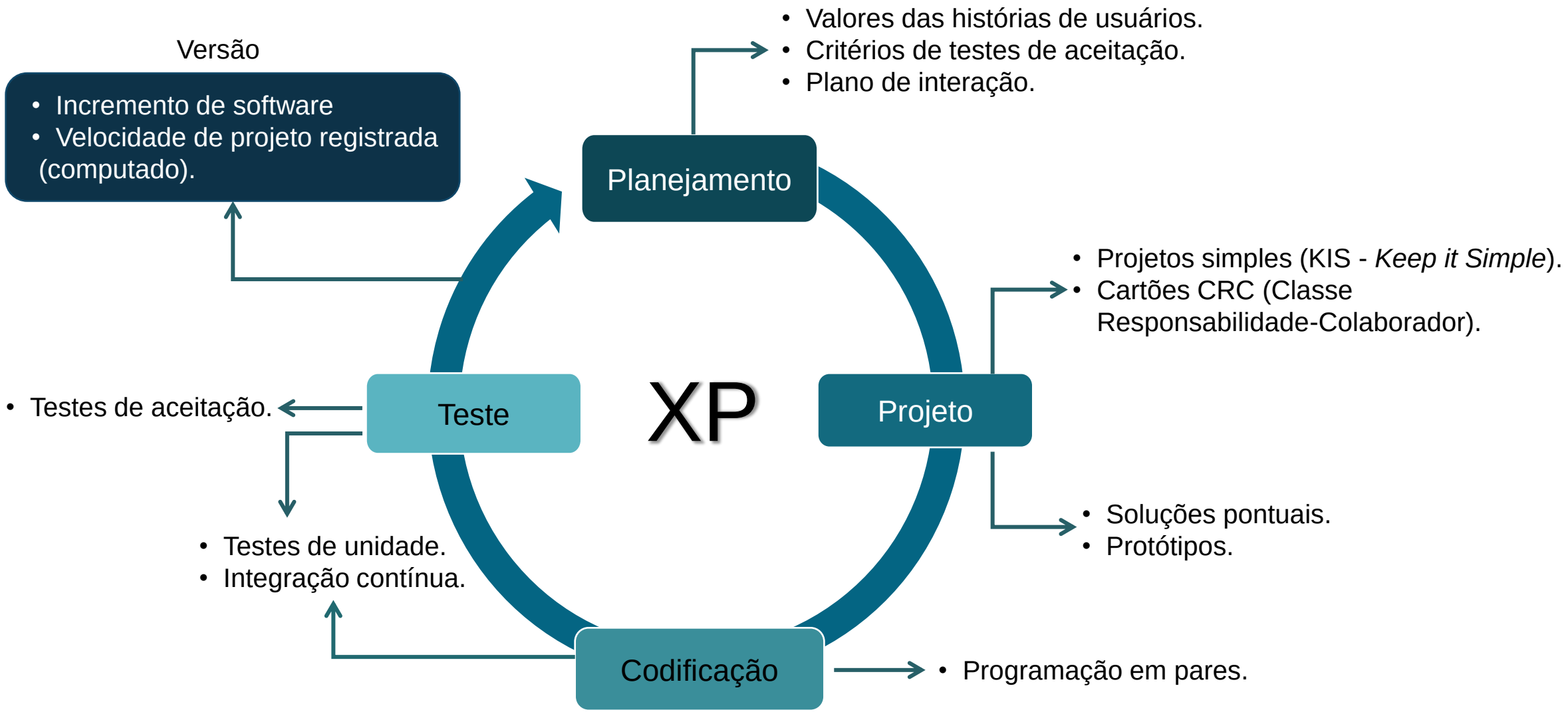
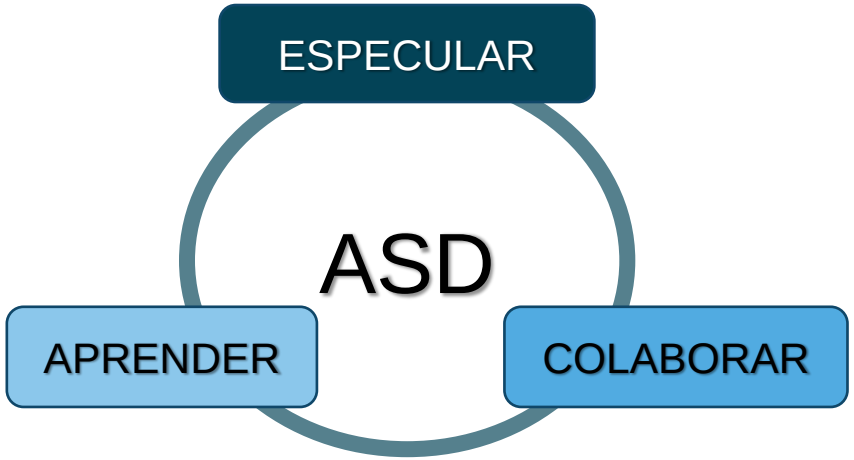


Figura 7. Processo *Extreme Programming* (XP). (PRESSMAN, 2011).

4.5. ASD (*Adaptive Software Development*)

- O *Adaptive Software Development* (ASD) é um método ágil que aplica ideias oriundas da área de sistemas adaptativos complexos (teoria do caos). (WAZLAWICK, 2013).
- Este modelo enxerga o processo de desenvolvimento de *software* como um sistema complexo com agentes (desenvolvedores, clientes e outros), ambientes (organizacional, tecnológico e de processo) e saídas emergentes (produtos sendo desenvolvidos).
- O modelo fundamenta-se em desenvolvimento cíclico iterativo baseado em três grandes fases.



- A ênfase global da ASD está na dinâmica das equipes auto-organizadas, na colaboração interpessoal e na aprendizagem individual e da equipe que levam as equipes de projeto de *software* a uma probabilidade muito maior de sucesso.

4.6. LSD (*Lean Software Development*)

- O desenvolvimento de *software* enxuto (*Lean Software Development*) adaptou os princípios da fabricação enxuta para o mundo da engenharia de software. (PRESSMAN, 2011). Os princípios enxutos que inspiraram o processo LSD podem ser sintetizados em:

Princípios LSD	
1.	Eliminar desperdício.
2.	Incorporar qualidade.
3.	Criar conhecimento.
4.	Adiar compromissos.
5.	Entregar rápido.
6.	Respeitar as pessoas.
7.	Otimizar o todo.

- De acordo com Pressman (2011), cada um dos princípios pode ser adaptado ao processo de *software*. Por exemplo, eliminar desperdício no contexto de um projeto de *software* ágil pode ser interpretado como:
    1. Não adicionar recursos ou funções estranhas.
    2. Avaliar o impacto do custo e do cronograma de qualquer requisito solicitado recentemente.
    3. Eliminar quaisquer etapas de processo supérfluas.
    4. Estabelecer mecanismos para aprimorar o modo pelo qual a equipe levanta informações.
    5. Assegurar-se de que os testes encontrem o maior número possíveis de erros.
    6. Reduzir o tempo necessário para solicitar e obter uma decisão que afete o *software* ou o processo aplicado para criá-lo.
    7. E racionalizar a maneira pela qual informações são transmitidas a todos envolvidos no processo.

5. BIBLIOGRAFIA

PRESSMAN, R. S. Engenharia de Software: Uma abordagem profissional. 7ª edição. Dados eletrônicos. Porto Alegre: AMGH-McGrawHill, 2011.

SOMMERVILLE, I. Engenharia de Software. 10ª edição. São Paulo: Pearson Prentice Hall, 2018.

WAZLAWICK, R. S. Engenharia de software: conceitos e práticas. Rio de Janeiro: Elsevier, 2013.


Obrigado!
Engenharia de Software