ENGENHARIA DE SOFTWARE

AULA 2

Prof. Alex Mateus Porn

CONVERSA INICIAL

Olá! Seja bem-vindo(a)!

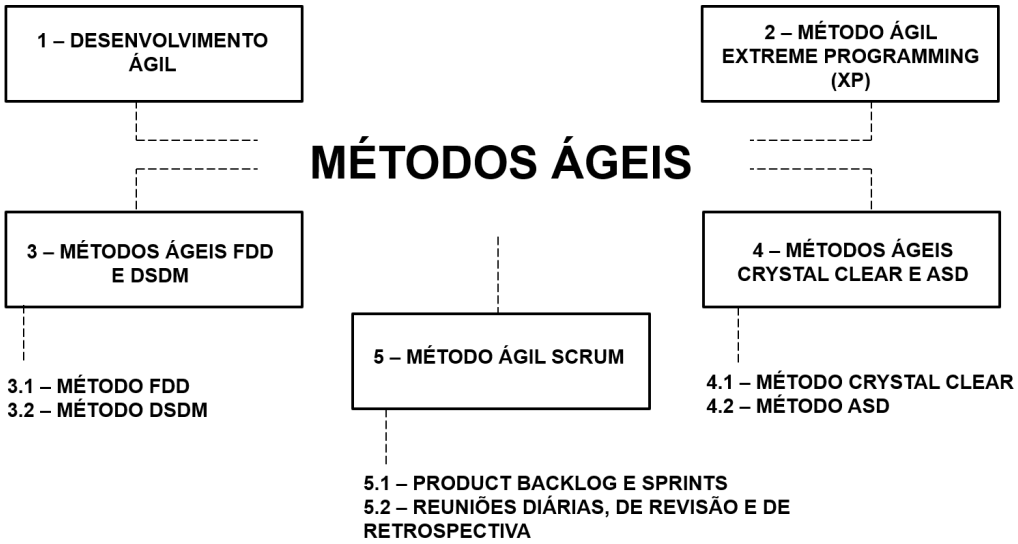
Nesta aula, abordaremos o conteúdo sobre os métodos ágeis de desenvolvimento de software, que oferecem metodologias de desenvolvimento mais rápidas e menos “burocráticas”, focando principalmente no desenvolvimento do sistema e menos em relatórios e documentos, porém, sem desprezá-los, com o objetivo de melhorar o processo de desenvolvimento frente aos modelos de processos prescritivos estudados anteriormente.

Esta aula tem como objetivo conhecer os principais métodos ágeis de desenvolvimento de software, as características de cada um, como são aplicados e possibilitar que possamos compará-los para decidir qual método utilizaremos futuramente em nossos projetos. Para isso, iniciaremos esta aula conceituando o que é desenvolvimento ágil e o que são e o que propõem os métodos ágeis.

Do mesmo modo como analisamos individualmente os principais modelos de processos prescritivos anteriormente, estudaremos individualmente os principais métodos ágeis nesta aula. Iniciaremos, no Tema 2, com o método XP, em seguida, no Tema 3, abordaremos os métodos FDD e DSDM, já no Tema 4, estudaremos os métodos *Crystal Clear* e ASD, e finalizaremos esta aula com o método Scrum, no Tema 5.

Portanto, ao longo desta aula, serão trabalhados os seguintes conteúdos:

Figura 1 – Métodos ágeis



TEMA 1 – DESENVOLVIMENTO ÁGIL

Os métodos ágeis se desenvolveram em um esforço para sanar fraquezas reais e perceptíveis da engenharia de software convencional. Conforme apontado por Cockburn (2002), os métodos ágeis se desenvolveram frente a várias falhas apresentadas pelos modelos prescritivos, como esquecer da fragilidade dos desenvolvedores e dos engenheiros de software, grande variação no estilo de trabalho, diferenças significativas no nível de habilidade.

No contexto da engenharia de software, Jacobson (2002) faz uma breve discussão sobre a agilidade no desenvolvimento de software:

> 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, mudanças de todos os tipos que poderão ter um impacto no produto que está em construção ou no projeto que cria o produto. Suporte para mudanças deve ser incorporado em tudo o que fazemos em software, algo que abraçamos porque é o coração e a alma do software. 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.

De acordo com Sommerville (2018, p. 60), quando modelos prescritivos são aplicados a sistemas de negócios de pequeno ou médio porte, a sobrecarga é tão grande que domina o processo de desenvolvimento de software. Mais tempo é investido na decisão de como o sistema deve ser desenvolvido do que na programação ou nos testes. À medida que os requisitos mudam, retrabalho é necessário e, ao menos a princípio, sua especificação e seu projeto têm de mudar.

Portanto, dadas as dificuldades inerentes dos métodos tradicionais de desenvolvimento de software, a insatisfação com essas abordagens levou ao surgimento dos métodos ágeis no final da década de 1990. Conforme apontado por Sommerville (2018) e Pressman (2011), a filosofia por trás dos métodos ágeis está refletida no *Manifesto para o Desenvolvimento Ágil de Software*, assinado em 2001 por Kent Beck e outros 16 renomados desenvolvedores, autores e consultores da área de software, batizados de *Agile Alliance*. Resumidamente esse manifesto propõe que estamos descobrindo maneiras melhores de desenvolver softwares, fazendo-os nós mesmos e ajudando outros a fazerem o mesmo. Por meio desse trabalho, passamos a valorizar:

- **Indivíduos e interações** mais que processos e ferramentas;
- **Software em funcionamento** mais que uma documentação abrangente;
- **Colaboração com o cliente** mais que negociação de contratos;
- **Responder a mudanças** mais que seguir um plano.

Nesse contexto, Pressman (2011, p. 83) define que a agilidade consiste em algo mais que uma resposta à mudança, abrangendo a filosofia proposta no manifesto ágil:

- Ela incentiva a estruturação e as atitudes em equipe que tornam a comunicação mais fácil entre membros da equipe;
- Enfatiza a entrega rápida do software operacional e diminui a importância dos artefatos intermediários;
- Assume o cliente como parte da equipe de desenvolvimento e trabalha para eliminar a atitude de “nós e eles”, que continua a invadir muitos projetos de software;
- Reconhece que o planejamento em um mundo incerto tem seus limites e que o plano de projeto deve ser flexível.

Com base nessa abordagem da necessidade da criação dos métodos ágeis, Pressman (2011, p. 83) ainda define que a agilidade pode ser aplicada a qualquer projeto de software. Porém, para obtê-la é essencial que:

- Seja projetada para que a equipe possa adaptar e alinhar tarefas;
- Possa conduzir o planejamento compreendendo a fluidez de uma abordagem de desenvolvimento ágil;
- Possa eliminar tudo, exceto os artefatos essenciais, conservando-os enxutos;
- Enfatize a estratégia de entrega incremental, conseguindo entregar ao cliente, o mais rápido possível, o software operacional para o tipo de produto e ambiente operacional.

Portanto, um processo ágil é caracterizado de uma forma que seja capaz de administrar a imprevisibilidade e seja adaptável. Parafraseando Pressman (2011), o processo ágil deve se adaptar incrementalmente com *feedbacks* constantes do cliente. Os incrementos do software, devem ser entregues em curtos períodos de tempo, de modo que as adaptações acompanhem o mesmo ritmo das mudanças. Essa abordagem iterativa capacita o cliente a avaliar o

incremento de software regularmente, fornecer o *feedback* necessário para a equipe de software e influenciar as adaptações de processo feitas para incluir adequadamente o *feedback*.

A *Agile Alliance*, por meio do *Manifesto Ágil*, estabelece 12 princípios de agilidade para quem quiser ter agilidade:

1. A maior prioridade é satisfazer o cliente por meio de entrega adiantada e contínua de software valioso;
2. Acolha bem os pedidos de alterações, mesmo atrasados no desenvolvimento. Os processos ágeis se aproveitam das mudanças como uma vantagem competitiva na relação com o cliente;
3. Entregue software em funcionamento frequentemente, de algumas semanas para alguns meses, dando preferência a intervalos mais curtos;
4. O pessoal comercial e os desenvolvedores devem trabalhar em conjunto diariamente ao longo de todo o projeto;
5. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e apoio necessários e confie neles para ter o trabalho feito;
6. O método mais eficiente e efetivo de transmitir informações para e dentro de uma equipe de desenvolvimento é uma conversa aberta, de forma presencial;
7. Software em funcionamento é a principal medida de progresso;
8. Os processos ágeis promovem desenvolvimento sustentável. Os proponentes, desenvolvedores e usuários devem estar capacitados para manter um ritmo constante indefinidamente;
9. Atenção contínua para com a excelência técnica e para com bons projetos aumenta a agilidade;
10. Simplicidade é essencial (a arte de maximizar o volume de trabalho não efetuado);
11. As melhores arquiteturas, requisitos e projetos emergem de equipes que se auto organizam;
12. A intervalos regulares, a equipe se avalia para ver como tornar-se mais eficiente, então sintoniza e ajusta seu comportamento de acordo.

De acordo com Pressman (2011, p. 85), nem todo modelo de processo ágil aplica esses 12 princípios com pesos iguais; alguns modelos ainda preferem ignorar a importância de um ou mais desses princípios. Entretanto, os princípios

definem um espírito ágil mantido em cada um dos modelos que veremos na sequência.

TEMA 2 – MÉTODO ÁGIL EXTREME PROGRAMMING – XP

O método ágil XP ou Programação Extrema, inicialmente, é adequado a equipes pequenas e médias. Esse método surgiu nos Estados Unidos no final da década de 1990. Conforme apresentado por Wazlawick (2013, p. 62), o XP preconiza mudanças incrementais e *feedback* rápido, além de considerar a mudança algo positivo, que deve ser entendido como parte do processo. Além disso, o XP valoriza o aspecto da qualidade, pois considera que pequenos ganhos a curto prazo pelo sacrifício da qualidade não são compensados pelas perdas a médio e a longo prazo.

Para aplicar o método XP, é necessário seguir uma série de práticas que dizem respeito ao relacionamento com o cliente, à gerência do projeto, à programação e aos testes (Wazlawick, 2013, p. 62):

a. **Jogo de planejamento:** semanalmente, a equipe deve se reunir com o cliente para priorizar as funcionalidades a serem desenvolvidas. Cabe ao cliente identificar as principais necessidades e à equipe de desenvolvimento estimar quais podem ser implementadas no ciclo semanal que se inicia. Ao final da semana, essas funcionalidades são entregues ao cliente. Esse tipo de modelo de relacionamento com o cliente é adaptativo, em oposição aos contratos rígidos usualmente estabelecidos.
b. **Metáfora:** é preciso conhecer a linguagem do cliente e seus significados. A equipe deve aprender a se comunicar com o cliente na linguagem que ele compreende.
c. **Equipe coesa:** o cliente faz parte da equipe de desenvolvimento e a equipe deve ser estruturada de forma que eventuais barreiras de comunicação sejam eliminadas.
d. **Reuniões em pé:** reuniões em pé tendem a ser mais objetivas e efetivas.
e. **Design simples:** implica atender a funcionalidade solicitada pelo cliente sem sofisticar desnecessariamente. 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. Nem sempre

o design simples é o mais fácil de se implementar, e o design fácil pode não atender às necessidades ou então gerar problemas de arquitetura.

f. **Versões pequenas:** 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.
g. **Ritmo sustentável:** trabalhar com qualidade um número razoável de horas por dia (não mais que oito). Horas extras só são recomendadas quando efetivamente trouxerem aumento de produtividade, mas não podem ser rotina.
h. **Posse coletiva:** o código não tem dono e não é necessário pedir permissão a ninguém para modificá-lo.
i. **Programação em pares:** a programação é sempre feita por duas pessoas em cada computador, em geral um programador mais experiente e um aprendiz. O aprendiz deve usar a máquina, enquanto o mais experiente deve ajudá-lo a evoluir em suas capacidades. Com isso, o código gerado terá sempre sido verificado por pelo menos duas pessoas, reduzindo drasticamente a possibilidade de erros.
j. **Padrões de codificação:** a equipe 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.
k. **Testes de aceitação:** são testes planejados e conduzidos pela equipe em conjunto com o cliente para verificar se os requisitos foram atendidos.
l. **Desenvolvimento orientado a teste:** antes de programar uma unidade, devem-se definir e implementar os testes pelos quais ela deverá passar.
m. **Refatoração:** não se deve fugir da refatoração quando ela for necessária. Ela permite manter a complexidade do código em um nível gerenciável, além de ser um investimento que traz benefícios em médio e longo prazo.
n. **Integração contínua:** 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.

Apesar de parecer ser uma vasta série de práticas a serem executadas na aplicação do método XP, essas práticas são condensadas basicamente em seis atividades, conforme podemos compreender melhor analisando a Figura 2 que demonstra o ciclo de lançamento (*release*) de um produto de software,

implementações, novas funcionalidades, entre outros requisitos solicitados pelo cliente, desenvolvidos seguindo a metodologia ágil XP.

Figura 2 – Ciclo de um *release* em XP

Fonte: Sommerville, 2018, p. 62.

Wazlawick (2013, p. 63) aponta como desvantagens do método XP, as práticas, porém, não são consenso entre os desenvolvedores, pois nem sempre são aplicáveis, o estilo de trabalho não é escalável para equipes maiores e que a programação em pares acaba sendo uma atividade altamente cansativa, que só é praticável se sua duração for mantida em períodos de tempo relativamente curtos.

Podemos colocar aqui também que o desenvolvimento em pares pode proporcionar um aumento no custo de desenvolvimento do software, em contrapartida pode proporcionar a redução de custos na fase de testes do sistema, considerando-se que dois programadores estão trabalhando em conjunto na mesma tarefa, assim como nos testes de aceitação.

Podemos fazer um comparativo das práticas de jogo de planejamento e reuniões em pé do método XP, com a metodologia de desenvolvimento Scrum, que estudaremos no Tema 5, a qual propõe a definição de *Sprints* de curtos períodos de tempo e reuniões diárias em pé de, no máximo, 15 minutos para tratar sobre as atividades atuais.

TEMA 3 – MÉTODOS ÁGEIS FEATURE DRIVEN-DEVELOPMENT – FDD E DYNAMIC SYSTEMS DEVELOPMENT METHOD – DSDM

Os métodos ágeis FDD e DSDM, apesar de serem métodos distintos, apresentam algumas características similares entre si, como ter o foco no desenvolvimento e, por funcionalidades, planejamento incremental e iterativo, integração contínua das funcionalidades e teste de software. Na sequência, veremos em detalhes as especificações de cada um desses dois métodos.

3.1 Método Feature Driven-Development – FDD

O Método FDD, ou Desenvolvimento Dirigido por Funcionalidade, é um método ágil que enfatiza o uso de orientação a objetos. Esse modelo foi apresentado em 1997, por Peter Coad e Jeff de Luca, como a evolução de um processo mais antigo (Wazlawick, 2013, p. 46).

Esse método é dividido em duas fases:

a. **Concepção e planejamento:** implica pensar um pouco (em geral de uma a duas semanas) antes de começar a construir o software.
b. **Construção:** desenvolvimento iterativo do produto em ciclos de uma a duas semanas.

Conforme apresentado por Wazlawick (2013, p. 47), a fase de concepção e planejamento possui três disciplinas (chamadas de *processos*):

a) **DMA – Desenvolver Modelo Abrangente:** em que se preconiza o uso da modelagem orientada a objetos.
b) **CLF – Construir Lista de Funcionalidades:** em que se pode aplicar a decomposição funcional para identificar as funcionalidades que o sistema deve disponibilizar.
c) **PPF – Planejar por Funcionalidade:** em que o planejamento dos ciclos iterativos é feito em função das funcionalidades identificadas.

Já a fase de construção incorpora duas disciplinas (processos):

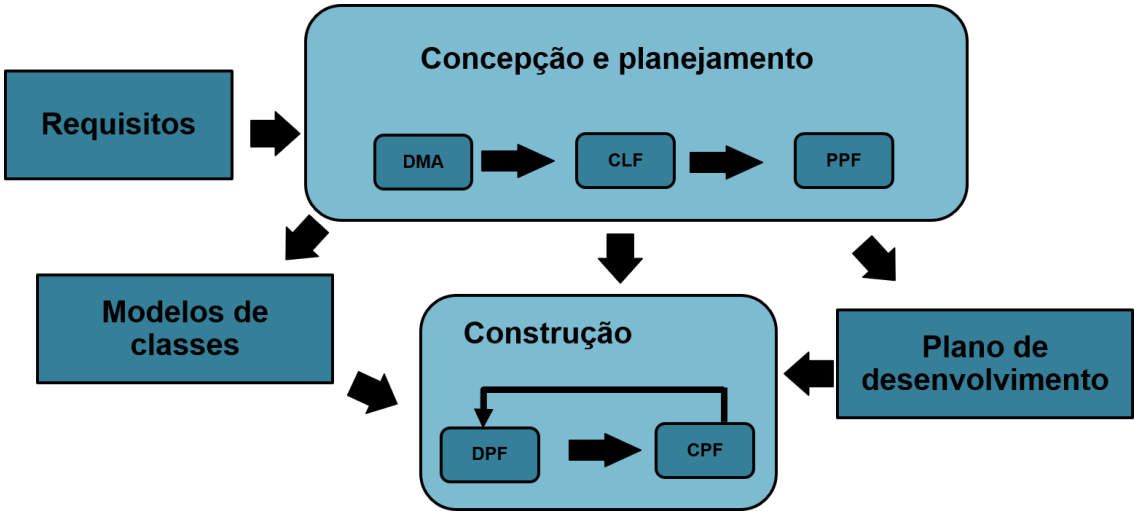
a) **DPF – Detalhar por Funcionalidade:** que corresponde a realizar o design orientado a objetos do sistema.
b) **CPF – Construir por Funcionalidade:** corresponde a construir e testar o software utilizando linguagem e técnica de teste orientadas a objetos.

Podemos também fazer uma comparação do método FDD com o método XP já estudado no tema anterior e com o método Scrum que veremos no Tema 5 desta aula. Podemos observar que as disciplinas Construir Lista de Funcionalidades – CLF e Planejar por Funcionalidades – PPF da fase de concepção, são muito similares a construção do "*Backlog* do Produto" do método Scrum, como veremos mais à frente.

Já a fase de construção do método FDD propõe uma metodologia muito semelhante a prática de Jogo de Planejamento do método XP e a definição dos *Sprints* no método Scrum, ou seja, a realização de atividades iterativas de curto prazo, com a finalidade de entregar ao cliente o software em partes funcionais.

Na Figura 3, podemos compreender de modo mais claro a aplicação dessas duas fases e seus processos.

Figura 3 – Ciclo de execução do método FDD



Fonte: Wazlawick, 2013, p. 47.

Podemos observar, pela Figura 3, que a fase de concepção e planejamento está estritamente relacionada à elicitação e à especificação dos requisitos, dos quais se originarão os modelos do software (exemplificado pelo modelo de classes na Figura 3) e os planos de desenvolvimento para a construção propriamente dito do sistema.

3.2 Método Dynamic Systems Development Method – DSDM

O método DSDM, ou Método de Desenvolvimento de Sistemas Dinâmicos, também é um modelo ágil baseado em desenvolvimento iterativo e incremental, com participação ativa do usuário (Wazlawick, 2013, p. 52).

Similar ao método FDD, o método DSDM é composto por 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.

Conforme apontado por Wazlawick (2013, p. 52):

> O DSDM fundamenta-se no Princípio de Pareto ou Princípio 80/20. 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.

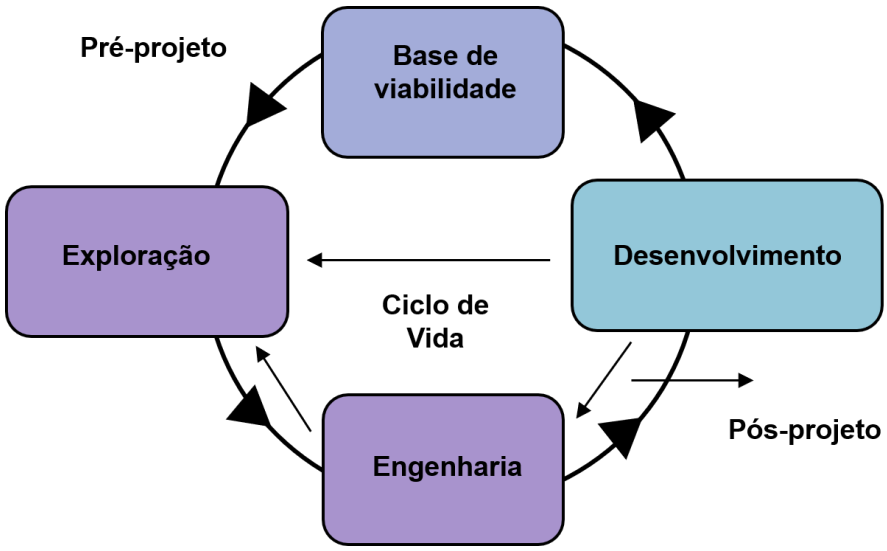
A filosofia DSDM pode ser resumida pelos seguintes princípios (Wazlawick, 2013, p. 52-53):

a) Envolvimento do usuário no projeto o tempo todo.
b) Autonomia dos desenvolvedores para tomar decisões sem a necessidade de consultar comitês superiores.
c) Entregas frequentes de *releases* suficientemente boas são o melhor caminho para conseguir entregar um bom produto no final. Postergar entregas aumenta o risco de o produto final não ser o que o cliente deseja.
d) Eficácia das entregas na solução de problemas de negócio. Como as primeiras entregas vão se concentrar nos requisitos mais importantes, elas serão mais eficazes para o negócio.
e) Feedback dos usuários como forma de realimentar o processo de desenvolvimento iterativo e incremental.
f) Reversibilidade de todas as ações realizadas durante o processo de desenvolvimento.

g) Previsibilidade para que o escopo e os objetivos das iterações sejam conhecidos antes do início.
h) Ausência de testes no escopo, já que o método considera a realização de testes como uma atividade fora do ciclo de vida.
i) Comunicação de boa qualidade entre todos os envolvidos é fundamental para o sucesso de qualquer projeto.

Na Figura 4, podemos ter uma melhor percepção da aplicação das três fases do modelo DSDM.

Figura 4 – Ciclo de desenvolvimento DSDM



Um ponto bastante interessante do método DSDM, em se tratando de um modelo iterativo, é que o método requer que cada etapa de desenvolvimento esteja suficientemente pronta para iniciar a próxima etapa, não precisando necessariamente, que a etapa anterior esteja completamente concluída para dar início ao próximo ciclo.

Na fase do Pré-Projeto, podemos observar as atividades de Estudo de Viabilidade e Exploração, com foco na análise das características do projeto, necessidades dos usuários e o levantamento de requisitos.

A fase do Ciclo de Vida tem início no Estudo de Viabilidade, porém, se concentra nas atividades de Exploração, Engenharia e Desenvolvimento, a qual é realizada em diversos ciclos iterativos que incorporam todos os princípios da filosofia DSDM.

Por fim, a fase de Pós-Projeto entra em ação após a conclusão de cada ciclo iterativo e entrega ao cliente, com objetivos de verificar se o software

funciona de forma correta e eficiente, realizar as atividades de manutenção e o aprimoramento do software se necessário.

TEMA 4 – MÉTODOS ÁGEIS CRYSTAL CLEAR E ADAPTIVE SOFTWARE DEVELOPMENT – ASD

Os métodos ágeis Crystal Clear e ASD foram criados muito próximos, sendo o Crystal Clear lançado em 1997 e o método ASD no ano 2000. Os dois métodos diferenciam-se basicamente na simplicidade das etapas propostas no método ASD em relação ao Crystal Clear, conforme veremos em mais detalhes nos tópicos a seguir.

4.1 Método Crystal Clear

O método Crystal Clear é um método ágil criado por Alistair Cockburn em 1997. Parafraseando Wazlawick (2013, p. 67), é uma abordagem ágil adequada a equipes pequenas (de no máximo oito pessoas) que trabalham juntas (na mesma sala ou em salas contíguas). Em geral, a equipe é composta por um designer líder e por mais dois a sete programadores. O método propõe, entre outras coisas, o uso de radiadores de informação, como quadros e murais à vista de todos, acesso fácil a especialistas de domínio, eliminação de distrações, cronograma de desenvolvimento e ajuste do método quando necessário.

O ciclo de vida do método Crystal Clear é organizado em três níveis:

1. **Iteração:** composta por estimação, desenvolvimento e celebração, que costuma durar poucas semanas.
2. **Entrega:** formada por várias iterações, que no espaço máximo de dois meses vai entregar funcionalidades úteis ao cliente.
3. **Projeto:** formado pelo conjunto de todas as entregas.

Conforme apresentado por Wazlawick (2013, p. 67-68), os sete pilares do método são listados a seguir:

1. **Entregas frequentes:** as entregas ao cliente devem acontecer no máximo a cada dois meses, com versões intermediárias.
2. **Melhoria reflexiva:** os membros da equipe devem discutir frequentemente se o projeto está no rumo certo e comunicar descobertas que possam impactar o projeto.

3. **Comunicação osmótica:** a equipe deve trabalhar em uma única sala para que uns possam ouvir a conversa dos outros e participar dela quando julgarem conveniente. Considera-se uma boa prática interferir no trabalho dos outros. O método propõe que os programadores trabalhem individualmente, mas bem próximos uns dos outros. Isso pode ser considerado um meio-termo entre a programação individual e a programação em pares, pois cada um tem a sua atribuição, mas todos podem se auxiliar mutuamente com frequência.
4. **Segurança pessoal:** os desenvolvedores devem ter certeza de que poderão falar sem medo de repreensões. Quando as pessoas não falam, suas fraquezas viram fraquezas da equipe.
5. **Foco:** espera-se que os membros da equipe tenham dois ou três tópicos de alta prioridade nos quais possam trabalhar tranquilamente, sem receber novas atribuições.
6. **Acesso fácil a especialistas:** especialistas de domínio, usuários e clientes devem estar disponíveis para colaborar com a equipe de desenvolvimento.
7. **Ambiente tecnologicamente rico:** o ambiente de desenvolvimento deve permitir testes automáticos, gerenciamento de configuração e integração frequente.

Podemos perceber algumas semelhanças do método Crystal Clear com outros métodos ágeis como o XP e, principalmente, o Scrum, quando podemos comparar a figura do designer líder ao papel do Scrum Master, e a utilização de quadros e murais ao uso do kanbam no Scrum, para especificar o que precisa ser feito, o que está em desenvolvimento e o que já foi feito.

4.2 Método Adaptive Software Development – ASD

O método ágil ASD, ou Desenvolvimento de Software Adaptativo, de acordo com Wazlawick, aplica ideias oriundas da área de sistemas adaptativos complexos. Esse método vê 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) (Wazlawick, 2013, p. 72).

O modelo fundamenta-se em desenvolvimento cíclico iterativo baseado em três grandes fases:

- Especular;
- Colaborar;
- Aprender.

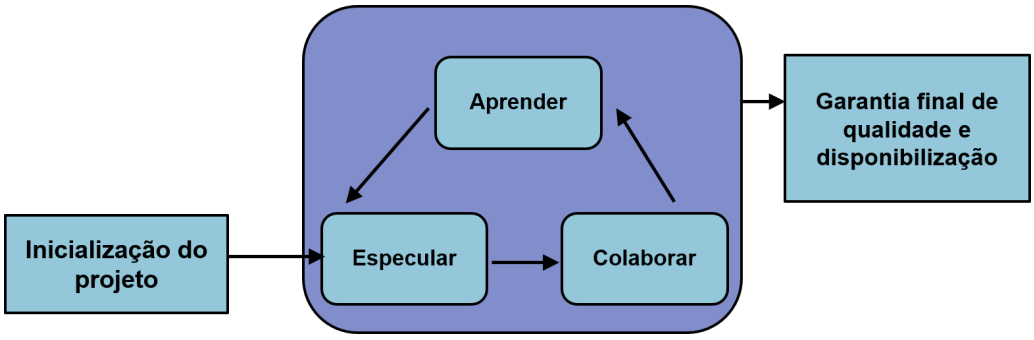
De acordo com essas três fases do modelo ASD, Pressman (2011, p. 94), afirma que:

> - **Na fase de especulação**, o projeto é iniciado e conduzido o planejamento de ciclos adaptáveis, com foco em determinar o tempo de duração do projeto, quantidade de ciclos e duração de cada um, objetivos de cada ciclo, componentes a serem desenvolvidos, tecnologias necessárias e listas de tarefas.
> - **Na fase de colaboração**, a equipe deve tentar equilibrar seus esforços para realizar as atividades que podem ser mais previsíveis e aquelas que são naturalmente mais incertas. A partir dessa colaboração, vários componentes serão desenvolvidos de forma concorrente.
> - **A fase de aprender**, corresponde a revisão de qualidade. Essa fase exige repetidas revisões de qualidade e testes de aceitação com a presença do cliente e de especialistas do domínio.

Com relação às três fases de execução do método ASD, podemos fazer uma leve comparação ao método DSDM. Se analisarmos a finalidade da fase de Especulação do método ASD, podemos observar a sua similaridade com a fase de Pré-Projeto do método DSDM. O mesmo ocorre entra a fase de Colaboração do ASD e o Ciclo de Vida do DSDM. Por fim, a fase de Aprender do método ASD, que envolve atividades de revisão de qualidade, testes e manutenção do sistema, é equivalente a fase de Pós-Projeto do método DSDM.

Na Figura 5, podemos compreender de modo mais simplificado como ocorre o ciclo de aplicação dessas três fases, desde o início do projeto até a sua entrega.

Figura 5 – Ciclo de execução ASD



Fonte: Wazlawick, 2013, p. 74.

Conforme observamos na Figura 5, mesmo havendo grande similaridade entre as fases dos métodos ASD e DSDM, ainda permanece a peculiaridade de cada um, principalmente na execução independente de cada fase no método DSDM e o ciclo iterativo entre as três fases do método ASD.

TEMA 5 – MÉTODO ÁGIL SCRUM

O método Scrum, conforme Wazlawick (2013), é um modelo ágil para a gestão de projetos de software. No Scrum, um dos conceitos mais importantes é o *sprint*, conforme já comentamos brevemente no início dessa aula, que consiste em um ciclo de desenvolvimento que, em geral, vai de duas semanas a um mês (Wazlawick, 2013, p. 56).

Pode-se dizer que o Scrum é atualmente o método ágil mais utilizado, principalmente porque pode ser integrado a outros métodos ágeis com facilidade, aplicando-se não só ao desenvolvimento de softwares como a qualquer ambiente de trabalho.

De uma forma bastante resumida, Sommerville (2011, p. 50) apresenta o Scrum em três fases:

> - **A primeira fase** corresponde a um planejamento geral, em que se estabelecem os objetivos gerais do projeto e da arquitetura do software;
> - **Na segunda fase**, ocorre uma série de ciclos de *Sprint*, sendo que cada ciclo desenvolve um incremento do sistema.
> - **Na terceira fase** encerra-se o projeto, completa-se a documentação exigida e, avaliam-se as lições aprendidas com o projeto.

Na execução dessas três fases apresentadas por Sommerville, há três papéis importantes no Método Scrum, conforme destacado por Wazlawick (2013, p. 56):

1. **Scrum Master:** é o responsável por manter o time Scrum em um ambiente propício para concluir o projeto com sucesso. Não é um gerente no sentido dos modelos prescritivos. Não é um líder, já que as equipes são auto-organizadas, sendo então um facilitador e um solucionador de conflitos.
2. ***Product Owner* (dono do produto):** é quem representa a voz do cliente, responsável por alcançar o maior valor de negócio para o projeto e

coordenação das necessidades dos clientes. 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*.

3. ***Scrum team* (Time Scrum):** é o responsável pelo desenvolvimento das entregas e entender os requisitos especificados pelo *Product Owner*. Trata-se da 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 seis a dez pessoas.

Além dos três principais perfis apontados por Wazlawick, podemos ainda destacar os seguintes papéis que podem fazer parte em um projeto gerenciado com o Scrum:

- ***Stakeholders* (partes interessadas):** são todos aqueles que tem algum tipo de interesse que o projeto exista ou funcione. Podem ser o cliente, usuários e patrocinadores.
- ***Scrum Guidance Body* (corpo de orientação do Scrum):** conjunto de documentos e/ou grupo de especialistas que estão envolvidos com a definição de objetivos relacionados com a qualidade, regulamentações governamentais de segurança e outros parâmetros-chave da organização. Auxilia na orientação dos trabalhos do *Product Owner*, *Scrum Master* e do Time Scrum.
- ***Chief Product Owner* (dono do produto chefe):** esse papel existe em projetos maiores, onde existem vários times Scrum.
- ***Chief Scrum Master* (*Scrum Master* chefe):** gerencia mais de um time Scrum.

5.1 *Product Backlog* e *Sprints*

Parafraseando Wazlawick (2013, p. 56), as funcionalidades a serem implementadas em cada projeto, como os requisitos por exemplo, são mantidas em uma lista chamada de *Product Backlog*.

Portanto, conforme Wazlawick (2013, p. 56), 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, para depois, à medida que o projeto avançar, tratar novas funcionalidades que forem sendo

descobertas. Isso, porém, não significa fazer um levantamento inicial excessivamente superficial. Deve-se tentar obter com o cliente o maior número possível de informações sobre suas necessidades. É possível que aquelas que efetivamente surgirem nessa interação tenham maior relevância do que outras que forem descobertas mais adiante.

A Tabela 1 apresenta um exemplo de um *Product Backlog* elaborado com base em Wazlawick (2013).

Tabela 1 – Exemplo de *Product Backlog*

Product Backlog					
Código	**Nome**	**Importância**	**Estimativa de esforço**	**Como demonstrar**	**Notas**
1	Depósito	30	5	Logar, abrir página de depósito, depositar R$ 10,00, ir para a página de saldo e verificar que ele aumentou em R$ 10,00.	Precisa de um Diagrama de Sequência UML. Não há necessidade de se preocupar com criptografia por enquanto.
2	Ver extrato	10	8	Logar, clicar em *Transações*. Fazer um depósito. Voltar para *Transações*, ver que o depósito apareceu.	Usar paginação para evitar consultas grandes ao BD. Design similar para visualizar da página de usuário.

No início de cada *Sprint*, que é o ciclo de desenvolvimento com poucas semanas de duração, 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, p. 57). De acordo com cada *Sprint*, Wazlawick (2013, p. 57) afirma que:

> Durante o *Sprint*, cabe ao *Product Owner* manter o *Sprint Backlog* atualizado, indicando as tarefas já concluídas e aquelas ainda por concluir, preferencialmente mostradas em um gráfico atualizado diariamente e à vista de todos. Tal gráfico pode se referir ao uso da técnica de Kanban.

5.2 Reuniões diárias, de revisão e de retrospectiva

Assim que um *Sprint* é iniciado, a cada 24 horas deve acontecer uma **Reunião Diária**, também conhecida como ***Stand Up Meeting*** **(Reunião em Pé)**, que tem como objetivo atualizar os membros do time sobre o andamento do *Sprint*.

Essa reunião deve ter duração limitada de 15 minutos, em que todos os membros do Time Scrum devem participar. Nenhuma das reuniões diárias deve ser cancelada ou adiada, mesmo com a falta de um ou mais membros. Cada reunião diária é organizada pelo Time Scrum, sendo o *Scrum Master* somente um facilitador para garantir o cumprimento do prazo e a aplicação das boas práticas.

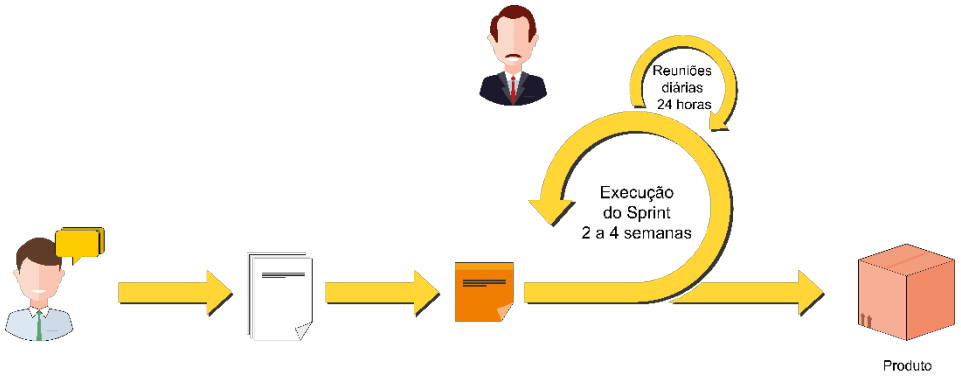
Cada membro do time deve responder a três questões em cada reunião diária:

1. O que eu fiz ontem?
2. O que farei hoje?
3. Quais impedimentos ou obstáculos estou enfrentando atualmente?

Outros assuntos, como discussões detalhadas, resolução de problemas ou elaboração de soluções de contorno devem ser tratadas em reunião específica, após a Reunião Diária, com os membros necessários.

Ao final de cada *Sprint*, deve haver uma **Reunião de Revisão** para avaliar o produto do trabalho, e, eventualmente, o *Sprint*, para avaliar os processos de trabalho. Assim, se aprovado, o produto (parcial ou final) poderá ser entregue ao cliente. Não sendo esse o *Sprint* final, o ciclo é reiniciado (Wazlawick, 2013, p. 57). A Figura 6 representa o processo de criação do *Product Backlog*, *Sprint Backlog* e realização das reuniões diárias.

Figura 6 – Ciclo Scrum



Créditos: Aa Amie/Shutterstock.

Cada Reunião de Revisão tem duração limitada de quatro horas. O Time Scrum apresenta, ou demonstra, as entregas do *Sprint* para o *Product Owner*, que avalia se atendem os requisitos propostos. Os itens não aceitos nessa reunião voltam para o *Product Backlog*. O *Scrum Master* é o responsável por garantir o bom andamento da reunião, que o *Product Owner* não altere os requisitos ou que rejeite um item porque quer alterá-los.

Finalmente, com o objetivo de analisar os *Sprints* concluídos e identificar possíveis melhorias para o processo, são realizadas as **Reuniões de Retrospectiva**. Cada reunião tem duração limitada de três horas. O Time Scrum conversa, avalia o que deu certo e errado, o que pode ser evitado e feito para reduzir erros, e identificar melhorias que podem resultar em simples acordos entre os membros para fazer as coisas de forma diferente e também na definição de regras formais. Trata-se de uma reunião em que a participação do *Product Owner* não é obrigatória.

FINALIZANDO

Nesta aula, pudemos compreender uma nova forma de gerenciamento de projetos de software, por meio dos métodos ágeis. Inicialmente, introduzimos os conceitos principais sobre desenvolvimento ágil para sistemas computacionais e uma abordagem sobre a necessidade do surgimento dos métodos ágeis e do manifesto ágil.

No decorrer do conteúdo, conhecemos seis diferentes métodos que podem ser aplicados para o gerenciamento de projetos de software. De acordo com o que vimos no manifesto ágil, são propostas 12 características para um

método ser considerado ágil. No entanto, é possível observar nos métodos que estudamos que nem sempre eles atendem a todas as 12 características, mas de acordo com a forma como são aplicados, considerando em sua metodologia o espírito ágil proposto pelo manifesto, caracterizam-se como tal.

Um ponto muito importante que pudemos observar nesta aula, assim como discutimos anteriormente, é que mais de um processo de software pode ser aplicado durante o desenvolvimento de um sistema, o mesmo ocorre com os métodos ágeis. Principalmente com o método Scrum, que estudamos no Tema 5. Vimos que é um dos métodos mais utilizados atualmente, principalmente por proporcionar sua aplicação em diferentes tipos de atividades.

Posteriormente, estudaremos sobre arquitetura de software, dando ênfase aos principais modelos de arquitetura existentes e suas necessidades de aplicação.

REFERËNCIAS

COCKBURN, A. **Agile Software Development**. Addison-Wesley, 2002.

JACOBSON, I. A Resounding Yes to Agile Processes: But Also More. **Cutter TI Journal**, v. 15, n. 1, p. 18-24, 2002.

SOMMERVILLE, I. **Engenharia de Software**. 10. ed. São Paulo: Pearson Education do Brasil, 2018.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011.

PRESSMAN, R. S. **Engenharia de Software:** uma abordagem profissional. 7. ed. Porto Alegre: AMGH, 2011.

WAZLAWICK, R. S. **Engenharia de Software:** Conceitos e Práticas. São Paulo: Elsevier, 2013.