ENGENHARIA DE *SOFTWARE*

Izabelly Soares de Morais

Metodologias ágeis de desenvolvimento

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Analisar o panorama e os conceitos no âmbito de desenvolvimento ágil.
- Descrever as práticas da modelagem ágil.
- Explicar os tipos e as etapas dos modelos de processos ágeis (Scrum, XP, DSDM).

Introdução

A agilidade é atualmente uma das habilidades mais expressivas, tanto em um indivíduo quanto em algum serviço prestado. Prestar um serviço para quem o utiliza é justamente o que um software faz, pois cada software é desenvolvido no intuito de solucionar problemas específicos.

A metodologia ágil visa trazer a agilidade para um contexto que antes era de ações tradicionais, tendo em vista que um processo de desenvolvimento de software é na verdade extremamente complexo; dessa forma, essas novas filosofias trazem consigo provavelmente algumas melhorias que ressaltam não só o produto, ou seja, o software, mas também a qualidade na interação entre a equipe de desenvolvimento e os clientes.

O contexto do desenvolvimento ágil

O termo desenvolvimento ágil se tornou bastante comum no universo da produção de software devido à alta demanda. Talvez você até se questione: mas como? Não foi sempre assim? Não, não foi sempre assim. Hoje, em razão do capitalismo e da evolução dos relacionamentos sociais entre as diversas nações existentes na sociedade moderna, os recursos, não só computacionais,

mas todos aqueles que auxiliam o homem em alguma atividade cotidiana, passaram a se tornar prioridade em todo processo de desenvolvimento.

Com isso, as atividades que os softwares devem desempenhar para que sejam considerados úteis estão se tornando cada mais robustas. Antes, um software realizava apenas funcionalidades básicas, como o processamento de uma soma com dois números, ou até mesmo o armazenamento de algum arquivo, o qual era bem simples, pois muitos nem imagens tinham, já que os recursos antigamente não suportavam tantos artefatos. A evolução disso tudo foi muito rápida, tanto que hoje é muito comum vermos o lançamento de um produto tecnológico que, depois de alguns meses, é lançado novamente com novas funcionalidades. Esses fatores evidenciam que, além de estarmos vivendo na era tecnológica, vivemos também na era da agilidade, e é justamente isso que o desenvolvimento ágil significa.

Conforme os relatos de Pressman e Maxim (2016, p. 66), em 2001, Kent Beck e outros 16 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" (*Manifest for Agile Software Development*). Ele declarava:

> Ao desenvolver e ajudar outros a desenvolver software, desvendamos formas melhores de desenvolvimento. Por meio deste trabalho passamos a valorizar:
> - Indivíduos e interações acima de processos e ferramentas
> - Software operacional acima de documentação completa
> - Colaboração dos clientes acima de negociação contratual
> - Respostas a mudanças acima de seguir um plano
>
> Ou seja, embora haja valor nos itens à direita, valorizaremos os da esquerda mais ainda.

Um dos pilares para o desenvolvimento ágil é justamente fazer uma junção dos aspectos relacionados ao desenvolvimento de um software, com a filosofia de que o cliente deve sair do processo satisfeito, e que isso, nesse contexto, acaba trazendo a aplicação de ações diversas, como a de que o cliente está sempre por dentro do que acontece na concepção de seu software, ou seja, em vez de o cliente ter o primeiro contato com o software ao final do processo, na metodologia ágil e em suas respectivas filosofias, o cliente terá acesso às partes do software conforme forem ficando prontas, fazendo parte também do processo.

Para Paula Filho (2009, p. 102), processos ágeis ou métodos ágeis constituem um grupo de metodologias diferentes entre si, mas caracterizadas por princípios comuns, mais baseados no trabalho cooperativo do que no formalismo e na documentação escrita. Para o autor, esses princípios foram reunidos por um grupo de metodologistas conhecidos no documento já mencionado chamado de Manifesto Ágil. Esse documento proclama os valores citados por Pressman e Maxim (2016), e que, aqui, Paula Filho (2009) traduz como os princípios que refletem nas atitudes de uma equipe de desenvolvimento:

- Satisfazer o cliente com entregas rápidas e frequentes de software útil.
- A funcionalidade é a principal medida de progresso.
- Mesmo alterações tardias de requisitos são bem-vindas.
- Colaboração próxima e diária entre gerentes e desenvolvedores.
- Comunicação face a face, facilitada pela colocalização.
- Ênfase na confiança em pessoas motivadas.

O que seria um processo caracterizado como ágil? Para Fowler (2005 apud Presmman, 2016, p. 69), devemos dar importância a três aspectos:

> 1. É difícil prever quais requisitos de software vão persistir e quais sofrerão alterações. É igualmente difícil prever de que maneira as prioridades do cliente sofrerão alterações conforme o projeto avança.
> 2. Para muitos tipos de software, o projeto e a construção são intercalados. Ou seja, ambas as atividades devem ser realizadas em conjunto para que os modelos de projeto sejam provados conforme são criados. É difícil prever quanto de trabalho de projeto será necessário antes que a sua construção (desenvolvimento) seja implementada para avaliar o projeto.
> 3. Análise, projeto, construção (desenvolvimento) e testes não são tão previsíveis (do ponto de vista de planejamento) quanto gostaríamos que fosse.

Um fator primordial do uso da metodologia ágil é a existência da adaptação muito rápida quando algo precisa ser mudado. Mas como isso pode acontecer, se todos os processos de desenvolvimento parecem bem complexos? E realmente são, mas, como a filosofia do método ágil consiste justamente em amenizar essa complexidade, talvez as mudanças não impactem tanto no resultado. O importante sempre é o prazo, e, claro, deixar o cliente satisfeito. A seguir, no próximo tópico, abordaremos mais detalhadamente do que consiste uma prática da metodologia ágil, ou seja, como ela se caracteriza.

Fique atento

Confira o que dizem Priklandnicki, Willi e Milani (2015, p. 16):

> Diversos profissionais reportam que o "pensamento ágil" já era adotado no desenvolvimento de software no Brasil bem antes da formalização do Manifesto Ágil. Por volta de 1999, Klaus Wuestefeld e Vinícius Teles, profissionais de desenvolvimento de software atualmente considerados precursores da agilidade no país, estavam em busca de uma abordagem alternativa que proporcionasse aumento das chances de sucesso no desenvolvimento de software. [...] Com a formalização do Manifesto, o movimento pela agilidade no Brasil teve início, em 2001, desencadeado por professores universitários e profissionais da indústria que tiveram contato com o movimento internacional, iniciado em 2000.

Práticas da metodologia ágil

Quando mencionamos o termo "práticas", estamos na verdade realizando alguns comparativos saudáveis em relação às metodologias tradicionais (citados às vezes como dirigidos por planos) e as ágeis. Essas práticas estão relacionadas a pontos cruciais em que uma metodologia se diferencia da outra, como documentações, equipes, processos de desenvolvimento, envolvimento do cliente, entre outras.

Segundo Pressman e Maxim (2016), a agilidade incentiva a estruturação e as atitudes em equipe que tornam a comunicação mais fácil (entre membros da equipe, entre o pessoal ligado à tecnologia e o pessoal da área comercial, entre os engenheiros de software e seus gerentes). Além disso, enfatiza a entrega rápida do software operacional e diminui a importância dos artefatos intermediários (nem sempre um bom negócio); aceita o cliente como parte da equipe de desenvolvimento e trabalha para eliminar a atitude de "nós e eles" que continua a impregnar muitos projetos de software; reconhece que o planejamento em um mundo incerto tem seus limites e que o plano (roteiro) de projeto deve ser flexível.

Um software é um produto, dessa forma, passa por um processo de desenvolvimento, como levantamento de requisitos, desenvolvimento, codificação, prototipagem, testes, entre outros. Com isso, todas as etapas demandam, além de tempo, investimentos financeiros.

Dentro desse âmbito, devemos sempre lembrar que tudo gera um custo, e que o valor final do trabalho está atrelado à metodologia e às ferramentas utilizadas. A Figura 1 traz uma demonstração da relação do custo de desenvolvimento com o cumprimento de um cronograma de desenvolvimento, diante das mudanças que podem surgir durante o processo.

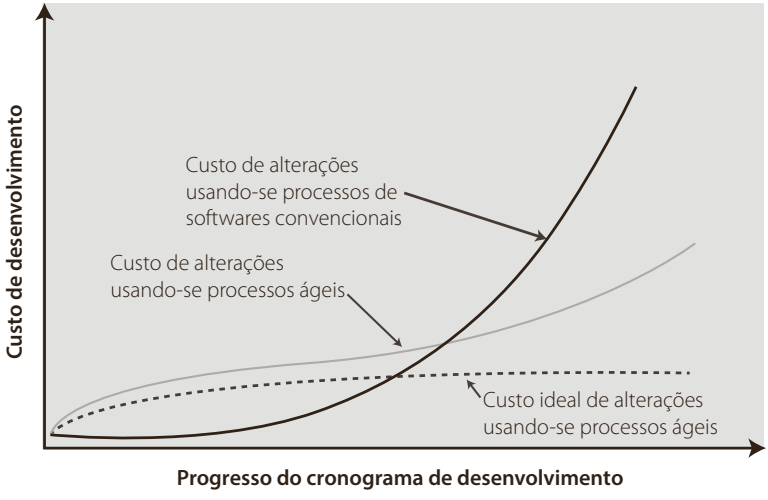

Figura 1. Custos de alterações como uma função do tempo em desenvolvimento.
Fonte: Pressman e Maxim (2016, p. 69).

Quando falamos em tradicional, não devemos pensar apenas em algo que não se aplicaria nos dias de hoje, a tão conhecida era moderna que vivenciamos atualmente. Nem tudo que carrega tradição é algo que deve ser taxado como retrógrado, pelo contrário, no caso das metodologias tradicionais de desenvolvimento de software, trazem bases de conhecimento suficiente para que a cada dia novas metodologias surjam. Já falamos sobre agilidade, e uma metodologia tradicional na verdade traz diversos passos e precauções, como na sua grande maioria uma vasta documentação relacionada ao software sendo desenvolvido. O Quadro 1 lista alguns aspectos importantes e suas distinções quando aplicado cada tipo de método.

Quadro 1. Métodos ágeis *versus* métodos dirigidos por planos

Área	Característica	Métodos ágeis	Métodos dirigidos por planos
Desenvolvedores	Cultura	Mais informal	Mais formal
	Tamanho da equipe	Menor	Maior
	Experiência	Maior	Menor
	Localização	Única	Espalhada
Clientes	Relacionamento	Mais cooperativa	Mais formal
	Localização	Próxima	Distante
Requisitos	Natureza	Emergentes	Bem conhecidos
	Variabilidade	Alta	Baixa
Arquitetura	Foco	Solução imediata	Expansibilidade
	Refatoramento	Fácil	Difícil
Projeto	Tamanho	Menor	Maior
	Objetivo	Valor rápido	Alta garantia

Fonte: Adaptado de Paula Filho (2009, p. 103).

Podemos notar que as áreas listadas por Paula Filho (2009) remetem a diversas características relevantes para cada área, e que cada metodologia se porta um pouco diferente da outra. Por exemplo, o envolvimento dos clientes, para a metodologia ágil, é muito importante, pois eles auxiliam a equipe na priorização dos requisitos. Por falar em equipe, nesse método, ela é valorizada conforme suas habilidades, e não precisam seguir métricas de trabalho, desde que consigam desenvolver as atividades às quais foram designadas.

Já falamos da aceitação de mudanças repentinas, e esse é um fato que deve ser ressaltado sempre. Desse modo, temos também uma característica bem

importante, que é a da entrega incremental, tipo de entrega que consiste em entrega de etapas de todo o processo ao cliente. Observe que, com esse tipo de ação, o cliente já fica a par de tudo que está acontecendo no desenvolvimento do software, sendo assim, pode até especificar requisitos para cada etapa. Pela descrição da figura, podemos observar que o projeto em si traz aspectos significantes, seja como a simplicidade de um todo, seja em relação as suas proporções.

Saiba mais

Conforme Ivar Jacobson, citado na obra de Presmman (2016, p. 68),

> Atualmente, agilidade se tornou a palavra da moda quando se descreve um processo de software moderno. Todo mundo é ágil. Uma equipe ágil é aquela rápida e capaz de responder de modo adequado às mudanças. Mudança tem tudo a ver com desenvolvimento de software. Mudança no software que está sendo criado, mudança nos membros da equipe, mudança 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 à mudança deve ser incorporado a 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.

Tipos de metodologias ágeis

No decorrer dos anos, assim como as necessidades sociais, conforme citamos anteriormente, acabaram acontecendo diversas evoluções, e houve até o surgimento de novas metodologias, e as ditas ágeis não poderiam ficar de fora. O Quadro 2 lista as principais metodologias ágeis e suas respectivas características mais relevantes.

Quadro 2. Comparação das principais características das metodologias ágeis

Metodologia	Criado por	Ano	Requisitos	Iterações	Incremental	Validação e teste	Diagramas da UML
XP	Kent Back	1996	Clientes escrevem	1–4 semanas com participação do cliente	Programação em duplas com *refactoring*	Teste de unidade e aceitação	Classes, caso de uso, sequência ou atividades
Scrum	Jeff Sutherland, Ken Schwaber e Mike Beedle	1995	*Product Backlog*	*Sprint* de 30 dias	*Sprint Backlog*	—	—
FDD	Jeff de Luca e Peter Coad	1997	Artefato ficha	2 semanas por prioridade	Refinamento	Inspeções pelos programadores	Classes, caso de uso, sequência e atividades
ASD	Sam Bayer e James Highsmith	1997	Sessões	4 a 8 semanas por quantidade de requisitos	Implementação das quantidades de requisitos	Testes funcionais	Classes, caso de uso, sequência e atividades

(Continua)

(Continuação)

Quadro 2. Comparação das principais características das metodologias ágeis

Metodologia	Criado por	Ano	Requisitos	Iterações	Incremental	Validação e teste	Diagramas da UML
CRYSTAL	Alistair Cockburn	1998	Entrevista, *briefing* e questionários	2 semanas	*Release* em sequência	Testes de regressão	Classes, caso de uso, sequência e atividades
ICONIX	Doug Rosenberg, Matt Stephens e Mark Colling Cope	1999	Entrevista, questionários e atas (rastreáveis)	2 semanas por prioridade	*Releases* em sequência	Testes funcionais, aceitação e de unidade	Classes, caso de uso, sequência ou atividades e robustez
DSDM	Consórcio de empresas associadas	1994	*Workshop* e estudo de viabilidade	2 semanas MOSCOW	*Timeboxing*	Testes de regressão	Classes, caso de uso, sequência e atividades
APSO	José Henrique T. C. Sbrocco e Paulo Cesar de Macedo	2011	Documento de visão, *Product Backlog*	1 semana (por orientação)	*Sprint Backlog*	Inspeções e testes funcionais	Classes, caso de uso, sequência ou atividades

Fonte: Adaptado de Sbrocco e Macedo (2012, p. 185).

Apesar de o quadro trazer uma síntese das principais metodologias, devemos ter ciência de que nem sempre elas serão aplicadas seguindo fielmente todos os aspectos de apenas uma metodologia, ou seja, pode ser que elas tenham sido citadas dessa forma e que em outro material difiram um pouco em relação à quantidade de iterações, entre outros. Além disso, é impossível abordarmos todos os assuntos de forma minuciosa aqui, pois esses conceitos englobam muito mais detalhes do que conseguiríamos citar. No entanto, vamos abordar a seguir as principais características de algumas das metodologias ágeis mais conhecidas, sem, é claro, desmerecer as outras.

Extreme Programming

Diante de uma grande quantidade de informações sobre a Extreme Programming (XP), ou, em português, Programação Extrema, podemos dizer que ela emprega uma metodologia orientada a objetos como paradigma de desenvolvimento e de quatro atividades metodológicas: planejamento, projeto, codificação, testes (PRESMMAN, 2016), conforme podemos observar na Figura 2.

Figura 2. Planejamento em Extreme Programming (XP).
Fonte: Sommerville (2011, p. 440).

A especificação de sistema em XP baseia-se em histórias de usuários que refletem as características que devem ser incluídas no sistema. No início do projeto, a equipe e o cliente tentam identificar um conjunto de histórias, o qual abrange toda funcionalidade que será incluída no sistema final. O próximo estágio é de estimativas, quando a equipe do projeto lê e discute as histórias e as classifica de acordo com o tempo que acredita ser necessário para implementá-las. O planejamento de release envolve a seleção e o refinamento das histórias que refletirão os recursos que serão implementados em um release de um sistema e a ordem em que as histórias devem ser implementadas; lembre-se de que o cliente deve estar envolvido nesse processo. De acordo com Sommerville (2011, p. 441):

> O planejamento de iteração é o primeiro estágio do processo de desenvolvimento de iteração. As histórias a serem implementadas para essa iteração são escolhidas, com o número de histórias que refletem o tempo para entregar uma iteração (normalmente, duas ou três semanas) e a velocidade da equipe.

As histórias do usuário podem ser acrescidas pelas metáforas, tendo em vista que elas são utilizadas com o intuito de trazer mais clareza diante do que está sendo solicitado pelo cliente, ou seja, para não haver dupla interpretação por parte da equipe. É uma ação que evita também que haja certa dependência de uma grande quantidade de documentação, o que não faz parte da metodologia XP. Inclusive o planejamento nessa metodologia acontece de forma incremental e interativa. As estimativas inseridas a partir das histórias do usuário podem ser classificadas como "baixa", "média baixa", "média alta" e "alta", podendo também receber valores de classificação.

Conforme Prikladnicki, Willi e Milani (2015), a Programação Extrema baseia-se em 14 princípios que funcionam como canais mentais para transformar os valores, que são abstratos, em práticas de fácil implementação no dia a dia do desenvolvimento. Os princípios são importantes porque ajudam a validar se novas práticas ou adaptações nas práticas originais estão alinhadas com o propósito da metodologia. O Quadro 3 apresenta os princípios propostos por Prikladnicki, Willi e Milani (2015).

Quadro 3. Princípios da Programação Extrema

Humanidade	A produção do software depende dos desenvolvedores. É importante levar em conta que suas necessidades individuais devem ser respeitadas e balanceadas com os interesses de negócio e as necessidades da equipe.
Economia	A equipe deve conhecer as necessidades de negócio e definir prioridades que agreguem o máximo de valor no menor intervalo de tempo. Flexibilidade para reagir a mudanças com rapidez é importante para acompanhar revisões nas prioridades de negócio.

(Continua)

(Continuação)

Quadro 3. Princípios da Programação Extrema

Melhoria	Melhorias devem ser implementadas constantemente. Primeiro, busque uma solução simples: isso satisfará o princípio da economia e ajudará a equipe a melhorar seu entendimento do problema. Depois, busque uma solução elegante e, por último, se isso ainda for prioritário, busque a solução ótima. Começar tentando a solução ótima vai custar mais tempo, e nem sempre ela será necessária.
Benefício mútuo	As atividades devem sempre trazer benefícios para os envolvidos. Testes, refatorações e metáforas são exemplos de atividades que trazem ganhos imediatos, pois aumentam a compreensão e a confiabilidade do software. Por outro lado, documentos extensos e planejamentos detalhados, de longo prazo, não trazem benefício imediato e estão altamente sujeitos a mudanças no futuro.
Semelhança	Boas soluções devem poder ser aplicadas novamente, inclusive em outros contextos e escalas. Devemos estar atentos para identificar estruturas equivalentes a padrões de projeto dentro do processo de desenvolvimento.
Diversidade	A equipe deve reunir muitas habilidades, opiniões e pontos de vista para aumentar sua flexibilidade e conseguir várias perspectivas que ajudarão a encontrar a melhor abordagem para cada situação. É importante que a equipe possua o valor do respeito para garantir que as diferenças criem um ambiente colaborativo que favoreça o crescimento de todos.
Passos pequenos	A integração do código, o desenvolvimento dirigido por testes e a entrega de novas versões devem manter um tamanho pequeno o suficiente para que a qualidade seja mantida.
Reflexão	Periodicamente, a equipe deve refletir sobre o próprio trabalho; assim poderá avaliar suas decisões e aprender com o passado, identificando pontos a melhorar e experiências de sucesso que podem ser repetidas.
Fluxo	O ritmo de trabalho deve ser sustentável ao longo do tempo, e a quantidade de software entregue também deve se manter estável. Para criar esse fluxo de entrega, a equipe deve ser capaz de integrar o código em períodos os mais curtos possíveis para minimizar o risco e o tamanho de prováveis defeitos.

(Continua)

(Continuação)

Quadro 3. Princípios da Programação Extrema

Oportunidade	A equipe deve estar sempre disposta a melhorar e atenta às oportunidades que surgem. Mudanças e problemas são vistos como meios para aprendizado e melhoria.
Redundância	Devemos acrescentar maneiras de assegurar a qualidade do software e reduzir os riscos do projeto como um todo. Aumentar as oportunidades de aprendizado diminui a concentração de conhecimento. Testar frequentemente reduz a chance de que defeitos sejam entregues ao cliente.
Falha	A equipe deve ter coragem para experimentar alternativas quando o seu caminho não estiver claro, mesmo sabendo que algumas dessas opções falharão. O mais importante é aproveitar essas oportunidades para adquirir aprendizado e estar disposto a abandoná-las para investir seus esforços na melhor opção.
Qualidade	A qualidade não é um fator negociável. O escopo e o tempo devem ser ajustados para entregar software de alta qualidade. O investimento na qualidade do código é importante para estabelecer o fluxo contínuo de entregas.
Aceitação da responsabilidade	As responsabilidades são aceitas, e não impostas. Cada membro deve estar comprometido e disposto a colaborar da melhor forma possível com a equipe.

Fonte: Adaptado de Prikladnicki et al. (2014, p. 45).

Outra característica relevante da XP é que há uma recomendação para que a codificação seja realizada em pares, na qual o objetivo é o de proporcionar a resolução do problema de forma mais rápida. Por exemplo, acredita-se que duas pessoas consigam solucionar o problema de forma mais ágil, tendo em vista que podem até discutir qual é a melhor solução para o problema que está sendo tratado, nessa ação pode ou não haver também a refatoração do código.

Os clientes estão constantemente realizando os chamados "testes de aceitação" ou "testes de cliente", nos quais, como já citado, se fazem presentes durante o processo de desenvolvimento. A documentação gerada pelo XP é na verdade o seu próprio código, assim como os testes realizados; dessa forma, há um acordo entre a equipe em trazer certa padronização ao código

para que simplifique o entendimento de todos a respeito do que está sendo desenvolvido. Devemos ter conhecimento também de que o código não possui um proprietário específico pois é armazenado em repositórios, e, na verdade, pertence a todos os envolvidos, visão que possibilita a todos realizar melhorias de codificação no que já foi desenvolvido.

Scrum

A metodologia Scrum é uma das mais peculiares, tendo em vista que suas métricas são famosas entre as demais metodologias. Conforme Prikladnicki, Willi e Milani (2015, p. 22-24):

> [...] o Scrum é um framework ágil que auxilia no gerenciamento de projetos complexos e no desenvolvimento de produtos. É conhecido como um framework que prescreve um conjunto de práticas leves e objetivas, muito utilizadas na área de desenvolvimento de software.

Ainda sob o ponto de vista dos autores, em projetos que executam o framework do Scrum, existem três papéis, três artefatos e quatro cerimônias:

- **Três papéis:** o Dono do Produto, responsável pelo valor de negócio do produto; o ScrumMaster, que serve aos outros papéis com processos e iniciativas para melhorar o trabalho; e a Equipe de Desenvolvimento, que se autoorganiza para entregar valor por meio do desenvolvimento de software com qualidade.
- **Três artefatos:** o Backlog do Produto, que lista a visão atualizada dos requisitos desejados para o produto; o Incremento do Produto, representado pelas funcionalidades de software pronto; e o Backlog da Sprint, que representa o planejamento estratégico e tático da próxima sprint em um nível mais micro.
- **Quatro cerimônias:** a reunião de planejamento da sprint, na qual será conhecida e planejada a meta de uma Sprint; as Scrum Diárias (Reuniões Diárias), que elevam o nível de auto-organização da Equipe de Desenvolvimento; a Revisão da Sprint, em que produto e projeto serão inspecionados e adaptados; e a Retrospectiva da Sprint, momento para inspeção e adaptação dos processos empíricos.

Na Figura 3, podemos visualizar os fluxos que um projeto, ao adotar a metodologia Scrum, deve seguir.

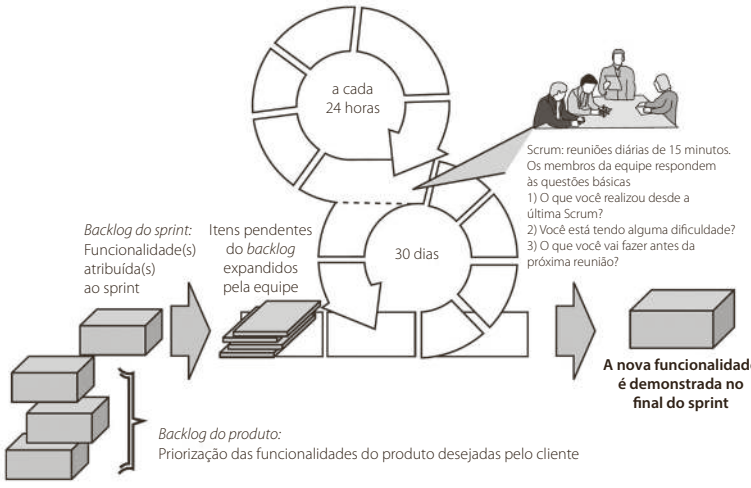

Figura 3. Fluxo do processo Scrum.

Fonte: Presmman (2016, p. 78).

No Scrum temos: *Backlog*, que é o nome dado à lista de tarefas a serem realizadas, as quais, na verdade, são os requisitos que vão compor as funcionalidades do sistema sendo desenvolvido. As chamadas sprints são ciclos de trabalho, ou seja, em uma sprint, há um *Product Backlog*, que é uma lista maior de requisitos, a qual é subdividida em *Sprint Backlog*, e que compõe a meta daquela sprint. Nesse mesmo tempo, há o ciclo de trabalho do Scrum, o qual compõe também uma reunião diária com todos os membros da equipe, na qual cada um diz o que fez, o que faltou fazer, o motivo e o que fará depois.

FDD

A história do Desenvolvimento Dirigido por Funcionalidades (FDD) é relatada por Prikladnicki, Willi e Milani (2015), que afirmam ter que se iniciado em Cingapura, entre os anos de 1997 e 1999. Um importante banco internacional, o United Overseas Bank (UOB), precisava de uma completa reengenharia de sua plataforma de empréstimos para corporações, comércio e consumidores, abrangendo os mais diversos instrumentos. Era um projeto complexo e o maior desse tipo na região. Após quase dois anos de consultoria, 3.500 páginas de casos de uso documentados, mas pouquíssimo software em execução, um dos

membros da equipe, Jeff De Luca, aceitou o desafio e convenceu a diretoria do banco a tentar mais uma vez, agora sob sua liderança e com uma pequena equipe de talentos de classe mundial para desenhar o sistema, treinar e agir como mentores de outros. Ele já adotara um processo leve e altamente iterativo em outros projetos e estava disposto a experimentar novas estratégias.

De acordo com relato descrito pelos autores, podemos notar que uma grande documentação foi criada e que quase nenhum software foi desenvolvido. Esse tipo de situação não é muito incomum, no entanto, apesar de a documentação ser importante, a produção tem de acompanhar todo o processo, pois, com certeza, sempre haverá um prazo de entrega final, e não adiantará entregar apenas documentação, já que não é isso que o cliente quer. Um fator que não podemos esquecer é que o cliente é leigo, e para ele o investimento que fez só será visto quando o software tiver em pleno funcionamento, e da forma como ele almejou.

Ainda conforme Prikladnicki, Willi e Milani (2015), FDD exibe uma notável característica social, refletida em sua proposta de estrutura da equipe de projeto. Os papéis principais que definem a estrutura fundamental de uma equipe FDD são:

- **Gerente de projeto —** É o responsável pelo gerenciamento geral administrativo do projeto, coordenando e alavancando as ações da equipe e reportando o progresso para a alta gerência, clientes e demais interessados.
- **Gerente de desenvolvimento —** Possui habilidades técnicas, gerenciais e de liderança necessárias para orquestrar as ações da equipe de desenvolvimento, operacionalizando as orientações fornecidas pelo gerente de projeto.
- **Especialistas no domínio de negócio —** Compreendem as regras e a dinâmica do domínio do negócio sendo considerado. São os responsáveis por informar e esclarecer à equipe do projeto o que deve ser feito para que o produto seja adequado às necessidades dos usuários. Pode incluir usuários e consultores externos.
- **Arquiteto líder —** Profissional com experiência em análise e modelagem de sistemas, capaz de liderar a equipe no desenvolvimento do modelo que será construído e refinado para implementar as funcionalidades identificadas.
- **Programadores líderes —** São os desenvolvedores mais experientes, reconhecidos como líderes pela equipe. Coordenam o desenvolvimento,

montam as equipes de funcionalidades e participam das definições técnicas, além de serem também proprietários de classes.
- **Proprietários de classes —** São os demais desenvolvedores da equipe, aos quais foram atribuídas certas classes do modelo.

DSDM

Metodologia de Desenvolvimento de Sistemas Dinâmicos (DSDM), sigla que vem do termo em inglês, *Dynamic Systems Development Method* e que em português significa Metodologia de Desenvolvimento de Sistemas Dinâmicos, segundo Pressman e Maxim (2016), é uma abordagem de desenvolvimento de software ágil que "oferece uma metodologia para construir e manter sistemas que satisfaçam restrições de prazo apertado por meio do uso da prototipação incremental em um ambiente de projeto controlado" (CS3, 2002 apud PRESSMAN; MAXIM, 2016). A filosofia DSDM baseia-se em uma versão modificada do princípio de Pareto — 80% de uma aplicação pode ser entregue em 20% do tempo que levaria para entregar a aplicação completa (100%).

O autor ainda fomenta que o DSDM Consortium (www.dsdm.org) é um grupo mundial de empresas-membro que assume coletivamente o papel de "mantenedor" do método. Esse consórcio definiu um modelo de processos ágeis, chamado ciclo de vida DSDM, que começa com um estudo de viabilidade, o qual estabelece os requisitos básicos e as restrições do negócio, e é seguido por um estudo do negócio, o qual identifica os requisitos de função e informação. Então, o DSDM define três diferentes ciclos iterativos (PRESSMAN; MAXIM, 2016):

- **Iteração de modelos funcionais —** Produz um conjunto de protótipos incrementais que demonstram funcionalidade para o cliente. (Observação: todos os protótipos DSDM são feitos com a intenção de que evoluam para a aplicação final entregue ao cliente.) Durante esse ciclo iterativo, o objetivo é reunir requisitos adicionais ao se obter feedback dos usuários, à medida que eles testam o protótipo.
- **Iteração de projeto e desenvolvimento —** Revê os protótipos desenvolvidos durante a iteração de modelos funcionais para assegurar-se de que cada um tenha passado por um processo de engenharia para capacitá-los a oferecer, aos usuários, valor de negócio em termos operacionais. Em alguns casos, a iteração de modelos funcionais e a iteração de projeto e desenvolvimento ocorrem ao mesmo tempo.

- **Implementação —** Coloca a última versão do incremento de software (um protótipo "operacionalizado") no ambiente operacional. Deve-se notar que: (1) o incremento pode não estar 100% completo ou (2) alterações podem vir a ser solicitadas conforme o incremento seja alocado. Em qualquer um dos casos, o trabalho de desenvolvimento do DSDM continua retornando-se à atividade de iteração do modelo funcional.

Link

O Manifesto Ágil foi o primeiro relato sobre a metodologia ágil e está disponível no seguinte link:

https://goo.gl/gxiQOn

Saiba mais

Exemplos de metodologias tradicionais de desenvolvimento de software: Cascata, de Prototipação, Espiral.

Referências

PAULA FILHO, W. P. *Engenharia de software*: fundamentos, métodos e padrões. 3. ed. Rio de Janeiro: LTC, 2009.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software*: uma abordagem profissional. 8. ed. Porto Alegre: AMGH; Bookman, 2016.

PRIKLANDNICKI, R.; WILLI, R.; MILANI, F. *Métodos ágeis para desenvolvimento de software*. Porto Alegre: Bookman, 2015.

SBROCCO, J. H. T. C.; MACEDO, P. C. *Metodologias ágeis*: engenharia de software sob medida. São Paulo: Érica, 2012.

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

Leitura recomendada

COHN, M. *Desenvolvimento de software com Scrum*: aplicando métodos ágeis com sucesso. Porto Alegre: Bookman, 2012. 496 p. (Livro eletrônico).

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo: