Programação Orientada e Objetos II

Anhanguera

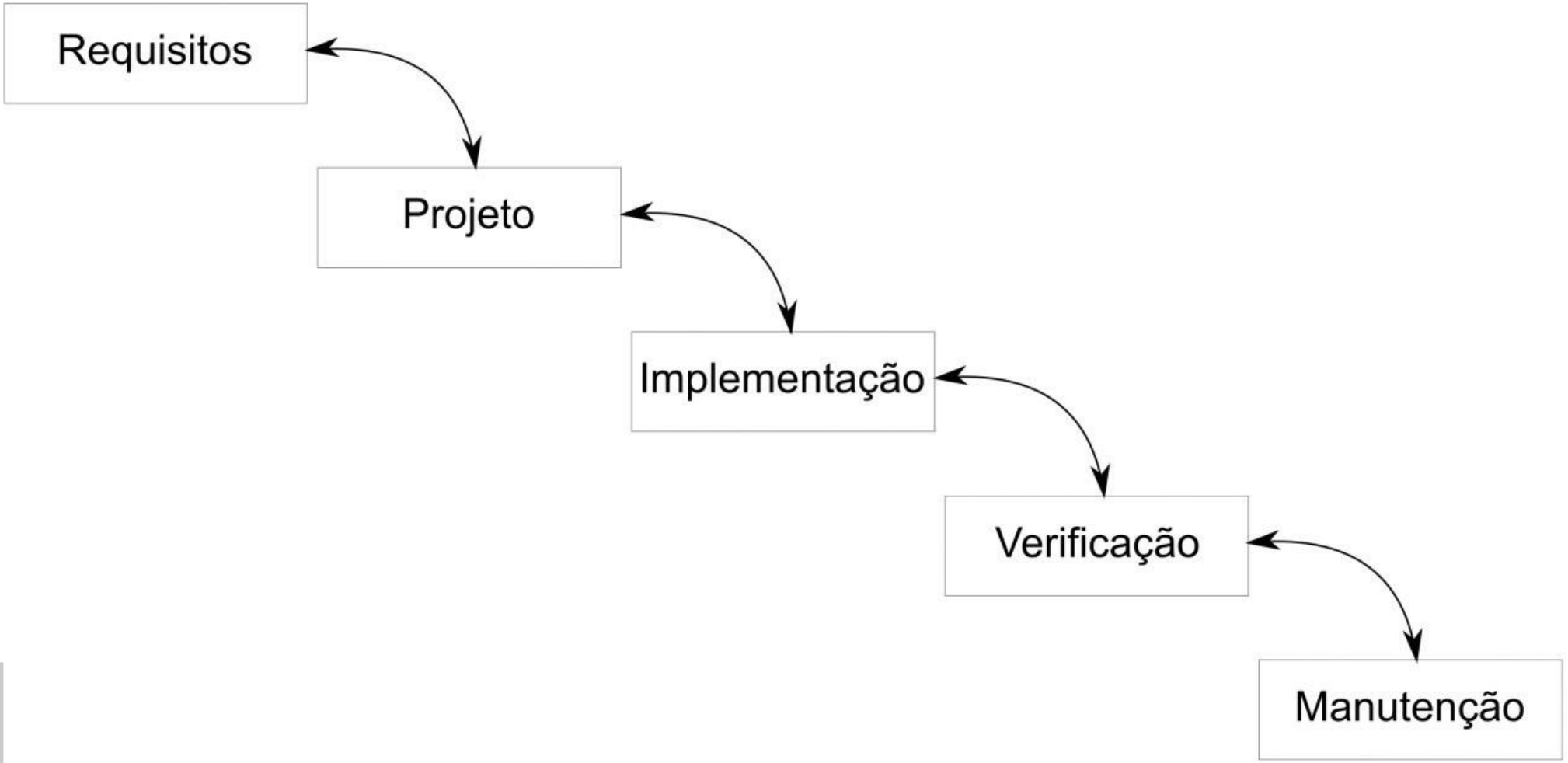

AVALIE
SUA PROFISSÃO
QUANDO APARECER EM SEU PORTAL UMA AVALIAÇÃO SOBRE SEU CURSO, RESPONDA:
NOTAS
9 OU 10
SIGNIFICA QUE VOCÊ INDICA
NOTAS
7 OU 8
SIGNIFICA QUE VOCÊ NÃO INDICA
Anhanguera

Anhanguera

Métodos ágeis em orientação a objetos

O desenvolvimento de software é essencial para todas as atividades, e podemos encontrar exemplos desta afirmação em todas as áreas, basta olharmos ao redor. Durante o início da computação, o software não recebia a sua devida importância, e o hardware recebia todo o crédito pelas aplicações computacionais. Contudo, esse contexto mudou rapidamente, devido à popularidade das soluções que o computador provia. A maneira de desenvolvimento de software seguia uma metodologia semelhante aos projetos de hardware, com formas rígidas de desenvolvimento, ignorando as necessidades de diversas partes. O modelo de cascata foi a primeira maneira de desenvolvimento e é considerado o modelo direto para o desenvolvimento de software. A Figura 3.8 apresenta as etapas clássicas de desenvolvimento de software nesse modelo (SOMMERVILLE, 2011):

1. Requisitos: levantar, entender e desenvolver as necessidades que levam à criação do software.
2. Projeto: elaborar os diagramas de Unified Modeling Language (UML - linguagem de modelagem unificada) e escolher as arquiteturas de software para o projeto.
3. Implementação: desenvolver o software utilizando os requisitos e o projeto feitos nas etapas anteriores.
4. Verificação: executar os testes no software e fazer a instalação.
5. Manutenção: verificar e corrigir problemas que foram detectados durante a operação do software.


Requisitos
Projeto
Implementação
Verificação
Manutenção

O modelo em cascata apresenta todas as etapas necessárias para que um software possa ser desenvolvido. Nesse cenário, se todas as etapas forem seguidas, o software estará pronto de acordo com o que foi pedido pelo cliente e com todo o planejamento elaborado pela etapa de projeto. Todavia, no decorrer do tempo os softwares podem apresentar diversos problemas, tais como (STELLMAN, 2015):

1. As etapas de requisitos e de projeto tomam muito tempo, comparadas ao projeto completo, assim, depois de certo tempo, apenas a documentação poderia ser entregue ao cliente, e não o software.
2. O software entregue, em diversos casos, não atende às necessidades do cliente, mesmo ele tenha as detalhado.
3. Durante o desenvolvimento, o projeto muda devido a dificuldades ou limitações, levando a uma implementação diferente do projeto.

4. Como o tempo de desenvolvimento pode ser grande, as etapas anteriores não geram entregáveis. Além disso, a etapa de testes pode ser negligenciada, validando os requisitos levantados no início do projeto, que podem não coincidir com o estado atual do desenvolvimento do sistema.

Devido às limitações que o modelo em cascata apresenta, surgiram outras formas de desenvolvimento de software. Esses modelos são caracterizados pela sua visão incremental dos estágios do modelo canônico cascata. Com isso, as etapas são feitas diversas vezes no projeto, até que o sistema esteja pronto. Dois modelos relacionados a esse tipo de abordagem são o em espiral e o de protótipo, que evitam diversos problemas que o modelo em cascata apresenta, tais como a entrega tardia do projeto e a distância entre requisitos e projeto com o software. Todavia, as próprias metodologias apresentam limitações em um aspecto essencial que os programas de computadores possuem, pois a maioria dessas aplicações é utilizada por pessoas que não são aquelas que fazem o pedido ou que estão envolvidas no processo. Ao final, mesmo com todo o aspecto incremental, as entregas continuam a enviar para o cliente algo que ele não esperava ou, ainda, algo em uma ordem não relacionada à importância para o cliente.

Diante desse cenário de diversas inconsistências, com o objetivo de encaminhar as formas de resolução para as questões de requisitos, prioridade de entrega, mudanças no projeto e outros, em 2001 foi elaborado um manifesto com 4 valores que devem serlevados em consideração durante o desenvolvimento de software(COHN, 2015):

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

Esses quatro valores são permeados por 12 princípios que os detalham, reforçando que as necessidades do cliente têm prioridade sob a metodologia. Nesse processo, as pessoas que fazem parte do uso, gerência, pedido e outros devem fazer parte do desenvolvimento, e as entregas devem ser feitas de forma constante e por prioridade que o cliente definir, mantendo a simplicidade do sistema.

O scrum é uma das diversas metodologias que usam conceitos ágeis, em que são definidos 3 papéis (STELLMAN, 2015):

1. Product owner (PO):

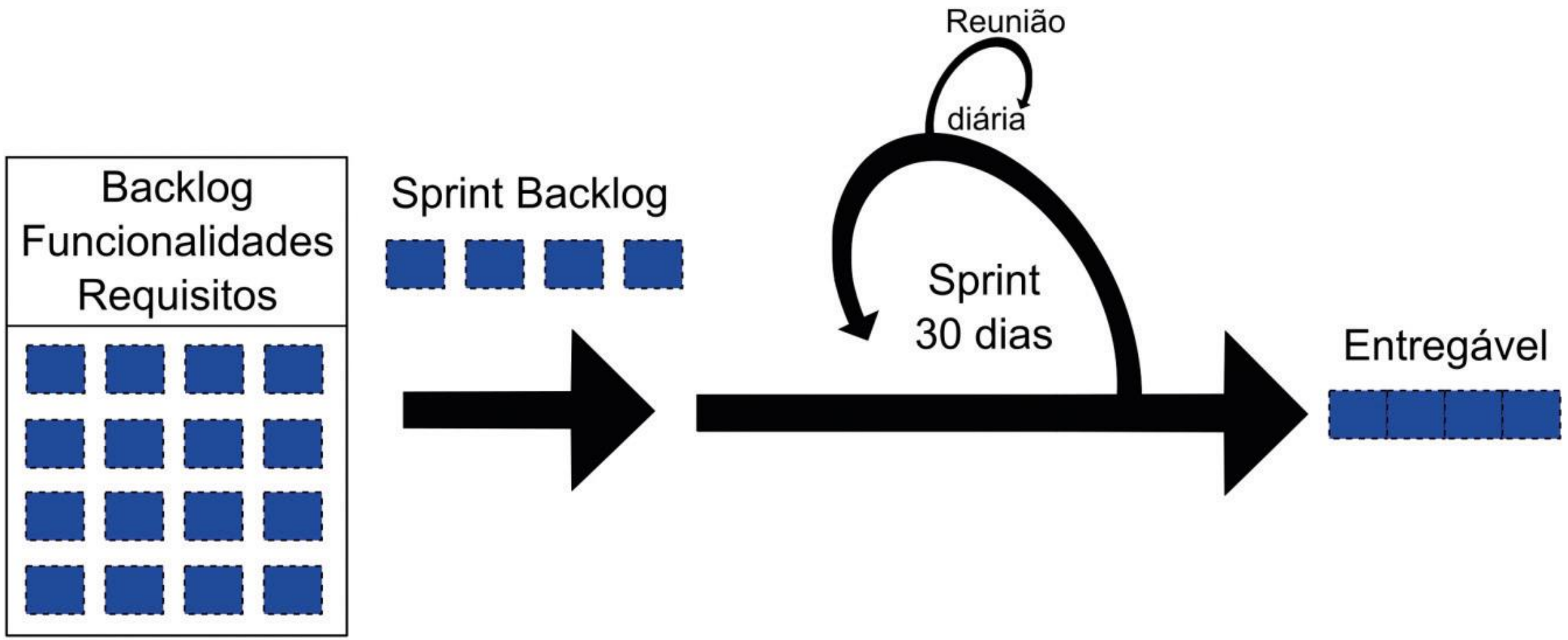
a. Pessoa responsável pelo produto a ser entregue.

b. Determina o que será desenvolvido e em qual ordem.

c. Faz a comunicação entre o time de desenvolvimento e as outras entidades para garantir a correta visão do que está sendo desenvolvido.

2. Scrum master:
a. Possui familiaridade pela etapa atual de desenvolvimento podendo ajudar toda equipe.
b. Ajuda a manter os princípios, valores e papéis do scrum.
c. Protege o time de desenvolvimento de interferências externas.
d. Resolve conflitos e problemas técnicos e pessoais para garantir a produtividade.
e. Não tem papel de gerência externa.
3. Time de desenvolvimento:
a. Grupo de pessoas que se auto-organiza para atingir o objetivo proposto pelo PO.
b. Com 5 a 9 pessoas, coletivamente possui todas as habilidades necessárias para realizar o projeto.

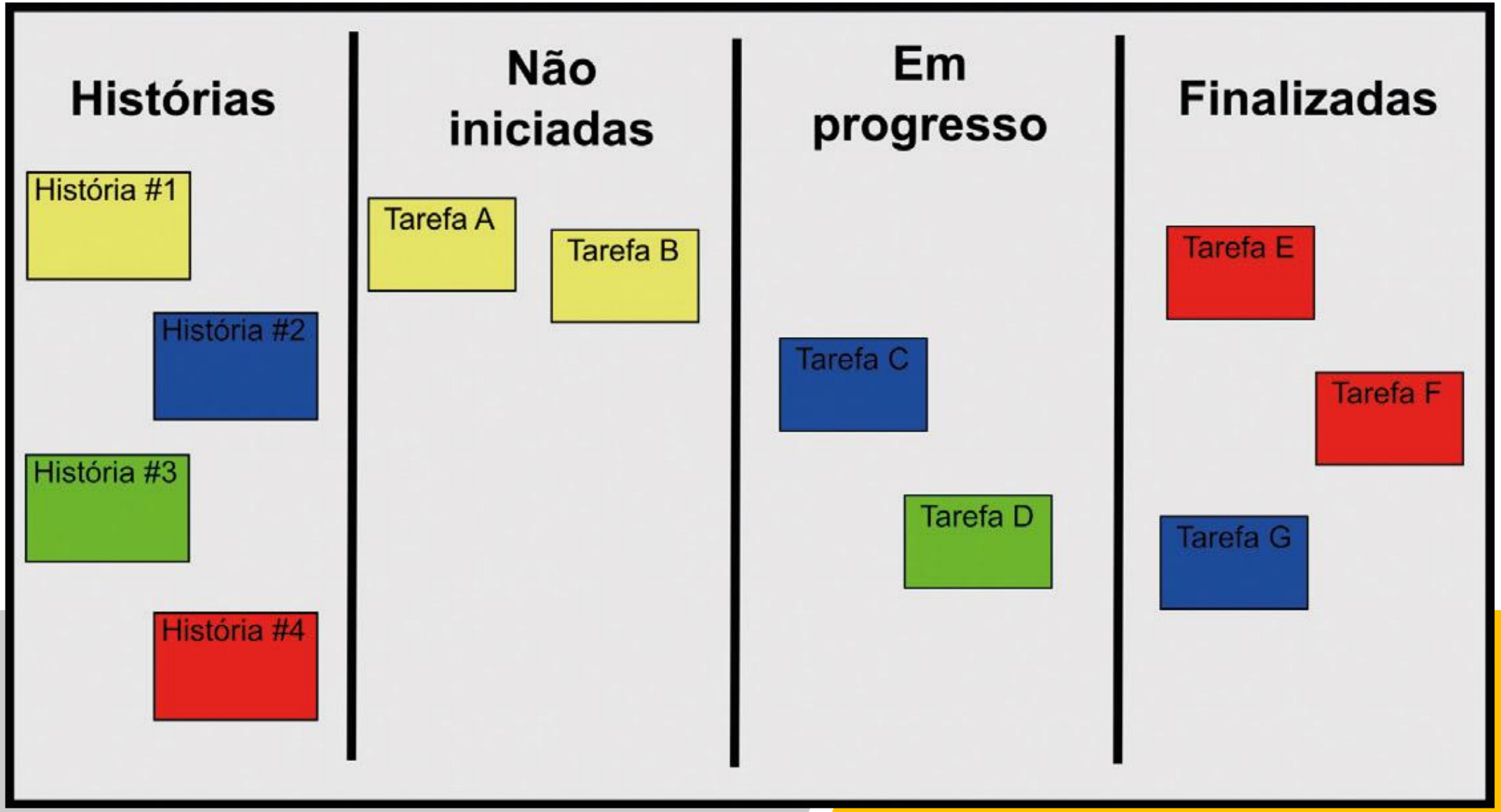
A Figura 3.9 apresenta a visão geral da metodologia scrum, em que o primeiro elemento a ser analisado é o backlog, que consiste em um conjunto de requisitos/funcionalidades elencadas pelo project onwer, junto com a equipe de desenvolvimento e o scrum master. A equipe de desenvolvimento pode opinar em relação aos requisitos, mas o PO tem prioridade no processo. O sprint backlog consiste em um conjunto de funcionalidades vindas do backlog, selecionadaspelo PO (com certa orientação técnica do scrum master e do time de desenvolvimento). Tais tarefas devem ser realizadas e entregues em certo período de tempo, chamado de sprint. Ao final de cada sprint é gerado um empregável para o cliente. Todos os dias são feitas reuniões com a equipe de desenvolvimento (daily meeting) para entender como está o desenvolvimento da sprint.


Reunião
diária
Backlog
Funcionalidades
Requisitos
Sprint Backlog
Sprint
30 dias
Entregável

Para criar o backlog são utilizadas histórias de usuários trazidas pelo PO, que contêm todas as necessidades do software. As histórias devem ter um formato específico, para aumentar as chances de o desenvolvimento ser correto:

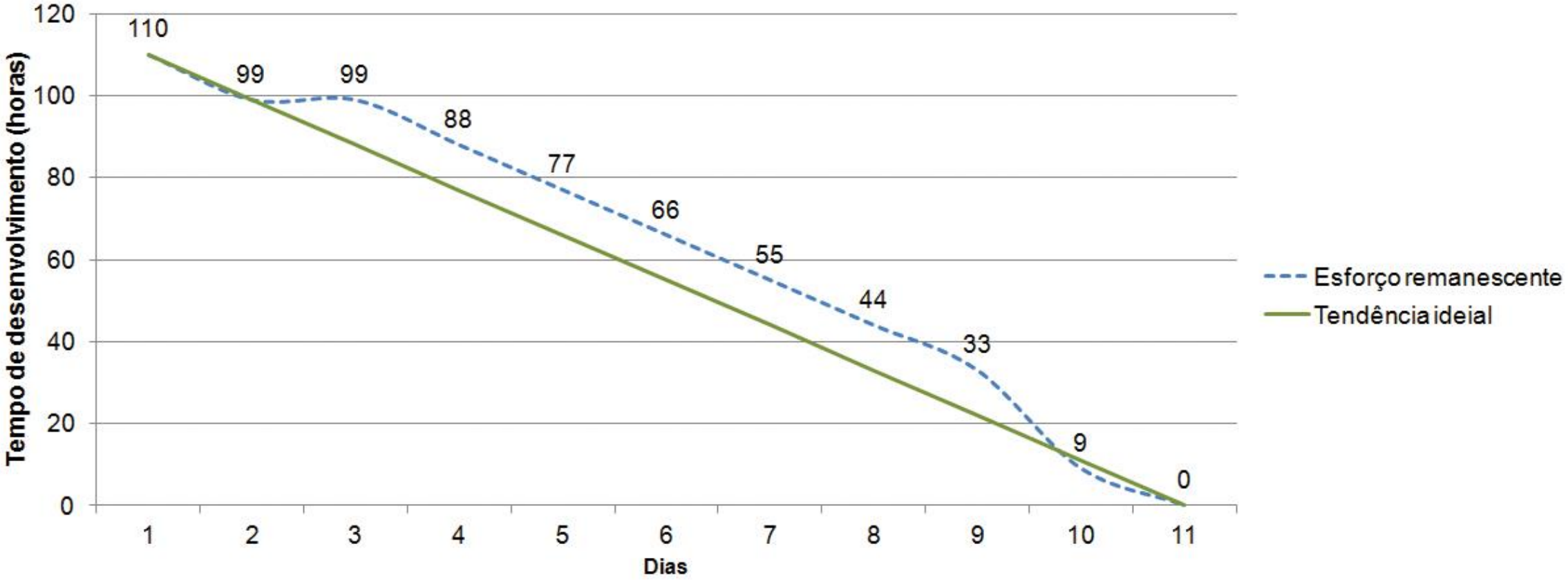
Como usuário <tipo de usuário>, eu preciso <objetivo>, pois <necessidade> O valor <tipo de usuário> deve explicar qual usuário fará a ação, por exemplo, o usuário padrão, administrador ou outros. O parâmetro <objetivo> descreve o que deve ser feito, contendo as suas necessidades e justificativas para implementação. O elemento que explica a razão auxilia no entendimento das necessidades daquela funcionalidade e em que momento deve ser priorizada.

Essas histórias são transformadas em tarefas, e o processo consiste em dividir as histórias em tarefas que possuem um nome e uma estimativa de tempo. Esse processo é muito importante, pois o tempo que se espera para concluir uma tarefa é utilizado para planejar o sprint. Essas tarefas levam em conta a criação das classes, pois estas serão criadas a partir dos substantivos das frases. Além disso, os atributos das classes poderão ser identificados pelos adjetivos e o relacionamento entre classes através dos verbos (DEITEL; DEITEL, 2016). A Figura 3.10 apresenta uma forma de organizar as tarefas. Pode ser utilizado um quadro branco ou até uma parede com blocos de papel adesivo para organizar as histórias de usuários, as tarefas iniciadas, as que estão em progresso e as que já foram finalizadas. Todas elas devem levar os conceitos da orientação a objetos e todas as formas corretas de planejamento.


Histórias
Não iniciadas
Em progresso
Finalizadas
História #1
História #2
História #3
História #4
Tarefa A
Tarefa B
Tarefa C
Tarefa D
Tarefa E
Tarefa F
Tarefa G

A atualização do quadro com as histórias e as tarefas que estão sendo feitas pode ocorrer tanto no momento em que chega nova demanda, quanto durante a sprint de desenvolvimento. Todos os dias, na reunião diária, as tarefas são atualizadas e todos informam o que foi feito. Além do quadro, é feito um gráfico burndown, que apresenta uma relação entre os dias da sprint e o esforço em horas da soma das tarefas, sendo T o tempo total de cada tarefa N e F o tempo utilizado para a tarefa N que foi efetivo no seu desenvolvimento. Isso mostra o tempo total ainda necessário para terminar a sprint.

A Figura 3.11 apresenta um exemplo desse gráfico, em que temos uma sprint de 10 dias, com uma estimativa de 110 horas de trabalho. Nos primeiros dois dias é possível reparar que o desenvolvimento estava indo como o planejado, pois a linha tracejada coincide com a linha solida, ou seja, a quantidade de horas planejadas dividida pelo número de dias da sprint está de acordo com o valor informado pela equipe do tempo que foi gasto nas etapas. Porém, no dia 3 o tempo reportado foi mais baixo, deixando horas a serem realizadas nos outros dias. Quase ao final da sprint foi possível realizar as tarefas atingindo o tempo necessário para terminar o desenvolvimento no prazo. Durante a sprint, o comprometimento é sempre com a entrega do que o PO apresentou como prioridade. Todavia, no backlog os desenvolvedores podem incluir elementos de documentação ou testes para garantir que as etapas consideradas sensíveis sejam feitas de maneira precisa.


120
100
80
60
40
20
0
Tempo de desenvolvimento (horas)
110
99
99
88
77
66
55
44
33
9
0
1
2
3
4
5
6
7
8
9
10
11
Dias
Esforço remanescente
Tendência ideial

As metodologias ágeis trouxeram fatores mais humanos para que o produto final seja realmente o que o cliente espera, tornando o desenvolvimento mais exato, assim, causando menos retrabalhos e menos atritos.

Mesmo com todos os aspectos incrementais, humanos e interativos que os métodos ágeis possuem, é sempre necessária a correção de problemas que um código-fonte pode ter. O sistema em seu início tem uma estrutura orientada a objetos, e os métodos são bem planejados para ter as características de coesão e acoplamento. Cada correção arruma os problemas, porém, tende por alterar a estrutura do software (COHN, 2015). Para resolver essa questão ou melhorar um código que foi mal elaborado, existe a refatoração. Esta etapa consiste em alterar a estrutura de um código sem modificar o seu comportamento, o que implica que seránecessário utilizar tempo no sprint para alterar o código que já foi implementado e está funcionando.

As razões que levam a essa demanda são diversas. Podemos citar alguns cenários que podem ser os sinais de que é necessário utilizar tempo para refatorar o código (MESZAROS, 2007):

- Nomes de variáveis, métodos ou atributos sem significado para o sistema.
- Partes do código duplicadas.
- Códigos extensos que poderiam ser reaproveitados de bibliotecas ou de outros projetos.
- Porções de códigos que não passaram nos testes (nesse caso, os testes unitários ganham destaque para forçar uma refatoração).
- O tempo gasto em correções dos bugs ser alto comparado ao tempo de desenvolvimento.
- Os programadores, quando têm que corrigir um problema em módulos de autoria de outras pessoas, utilizam muito mais tempo que nos de própria autoria.

Todos esses sintomas devem ser avaliados durante as etapas de desenvolvimento para desencadear os processos de refatoração. Existem momentos indicados para que seja feito o processo de correção do código, que podem ser:

- Quando se adiciona uma nova característica ao software, quando é possível ver quais partes são utilizadas e, então, corrigir os problemas.
- No próprio processo de correção de bugs pode-se averiguar as condições do código e aplicar as correções.
- Em casos mais complexos, pode ser necessário fazer uma revisão de código e aplicar a refatoração, dessa forma, tenta se minimizar o tempo de correção e entendimento do código.

Os processos de refatoração podem ser descritos como uma revisão e melhor entendimento de cada parte do código. Há metodologias precisas para descrever como um código deve ser refeito para se tornar melhor. Aqui abordaremos as formas mais gerais:

- Métodos devem ser claros, e a sua função precisa ser muito bem descrita. A função de um método deve ser uma tarefa direta e simples.
- As classes necessitam sempre buscar a alta coesão e o baixo acoplamento, ou seja, fazer ações específicas de maneira independente.
- Durante a refatoração não se deve incluir novas funções, o foco deve ser deixar o código mais legível e preciso.

- Todos os testes ao final da refatoração devem ter sucesso, e essa é uma medida de qualidade essencial em projetos.

A refatoração consiste em observar o código e avaliar se é necessário melhorar a base já implementada para que o projeto tenha uma duração longa. Levando em conta os recursos vistos nessa seção, você aumenta as chances de sucesso do seu projeto!

O desenvolvimento de software sofreu grandes mudanças no decorrer do tempo. No princípio, cada etapa era muito bem definida, levando à geração da documentação completa antes da produção de código e outros efeitos. Com isso, todo o processo de desenvolvimento transcorria sem a interação com o cliente, criando, assim, casos em que a entrega final não era o que cliente esperava.

Observe as afirmações abaixo:

I - Indivíduos e interações mais que processos e ferramentas.

II - Software em funcionamento mais que documentação abrangente.

III - Colaboração com o cliente mais que negociação de contratos.

IV - Responder a mudanças mais que seguir um plano.

V - Elementos de programação mais que processos de controle.

Quais delas pertencem aos valores do manifesto ágil?

a) Apenas as afirmações I, II e V.

b) Apenas as afirmações I, II, III e V.

c) Apenas as afirmações I, II, III e IV.

d) Apenas as afirmações II e III.

e) As afirmações I, II, III, IV e V.

Para o levantamento de requisitos, há diversas formas de buscar o entendimento do que o cliente procura no software solicitado. Uma delas é a história de usuários, que são pequenos textos que refletem o que deve ser feito, por quem e qual é a razão.

Durante o desenvolvimento de um software utilizando o scrum, onde são aplicadas as histórias de usuário?

a) As histórias de usuário são aplicadas durante a sprint para verificar qual é o modelo de teste necessário.

b) As histórias de usuário são usadas apenas na etapa de planejamento.

c) As histórias de usuário são utilizadas apenas para verificar se a sprint deve ter mais ou menos que 30 dias.

d) As histórias de usuário são utilizadas para montar o product backlog e, com isso, criar as tarefas da sprint.

e) As histórias têm papel figurativo, servindo apenas para ilustrar um cenário já consolidado.