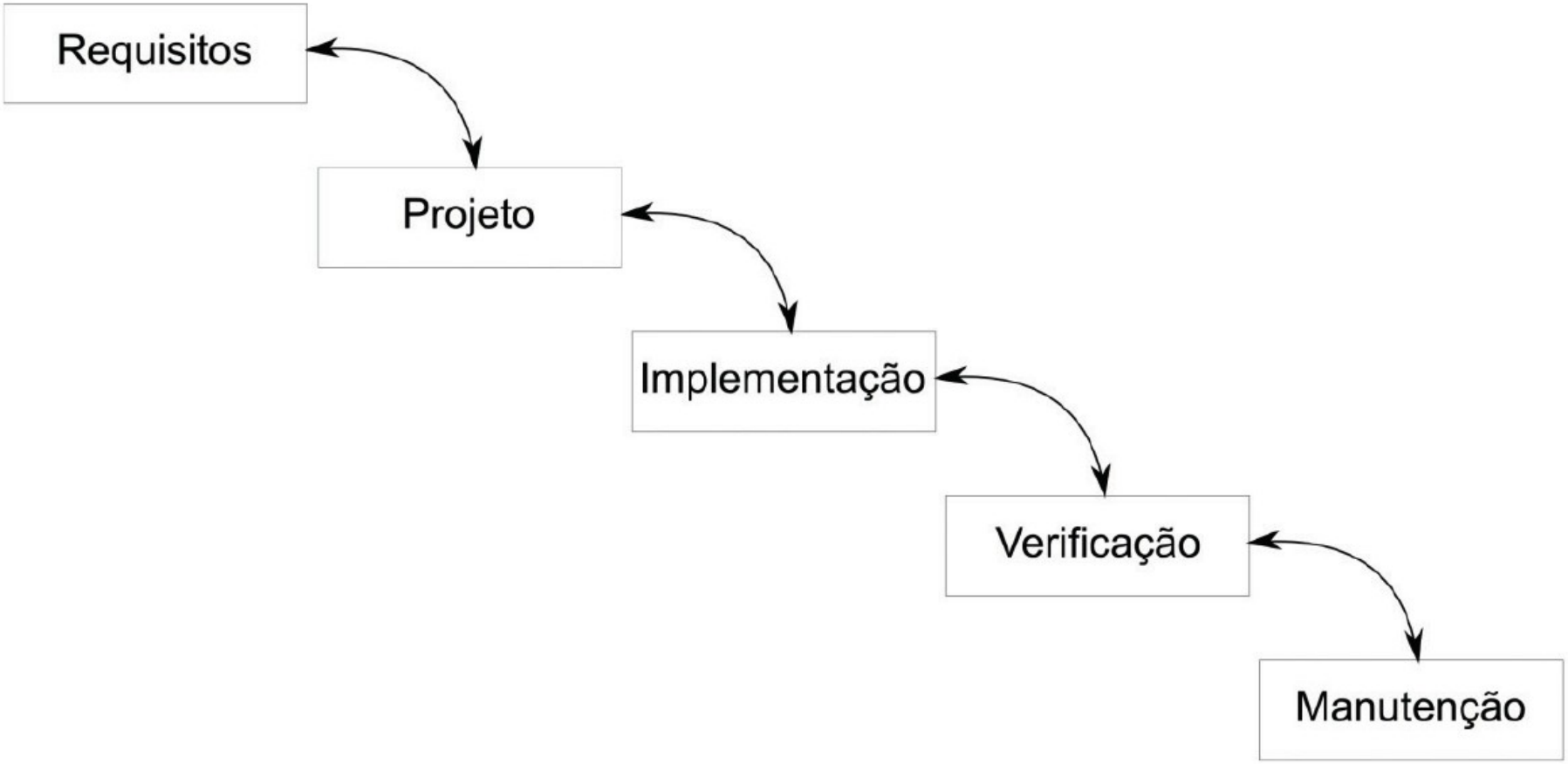
Métodos ágeis em orientação a objetos

Programação Orientada a Objetos

2023-05-02

Modelo de cascata

Figura 3.8 | Modelo clássico cascata de desenvolvimento de software



- Primeira maneira de desenvolvimento
- Considerado o modelo direto para o desenvolvimento de software

Modelo de cascata

- Requisitos
  - Levantar, entender e desenvolver as necessidades que levam à criação do software
- Projeto
  - Elaborar os diagramas UML
  - Escolher as arquiteturas de software
- Implementação
  - Desenvolver o software utilizando os requisitos e o projeto feitos nas etapas anteriores

Modelo de cascata

- Verificação
  - Executar os testes no software e fazer a instalação
- Manutenção
  - Verificar e corrigir problemas que foram detectados durante a operação do software

- O modelo em cascata apresenta todas as etapas necessárias para que um software possa ser desenvolvido
- No decorrer do tempo podem surgir problemas
- As etapas de requisitos e projeto tomam muito tempo, comparadas ao projeto completo
  - Antes de começar a implementação
  - Neste período, apenas a documentação seria entregue ao cliente, e não o software ou parte dele

- Em algum casos o software entregue não atende às necessidades do cliente, mesmo que ele tenha detalhado
- O projeto muda durante o desenvolvimento devido a dificuldades, limitações, etc.
- Como o tempo de desenvolvimento pode ser grande, as etapas anteriores não geram entregáveis (software)
- Problemas na etapa de testes
  - Ex.: validação de requisitos desatualizados
    - Mudaram após a etapa de projeto

Outros modelos

- São caracterizados por uma visão incremental
  - “Fazer por partes e ir avançando”
- Etapas são repetidas até que o sistema esteja pronto
- 2 exemplos de modelos
  - Em espiral
  - De protótipo
- Tentam evitar
  - Entrega tardia do projeto
  - Distância entre projeto e software

- As metodologias apresentam limitações em um aspecto essencial que os programas de computadores possuem
- As aplicações são utilizadas por pessoas que não são aquelas que fazem o pedido
- Mesmo com todo o aspecto incremental, as entregas continuam a enviar para o cliente algo
  - Que ele não esperava
  - Em uma ordem não relacionada à importância

Manifesto Ágil

- Publicado em 2001
- Trabalho de 17 desenvolvedores de software interessados em buscar uma alternativa aos atuais processos de desenvolvimento de software
  - 4 valores detalhados em 12 princípios
  - O cliente tem prioridade

4 valores

1. Indivíduos e interações mais que processos e ferramentas

2. Software funcionando mais que documentação abrangente

3. Colaboração com o cliente mais que negociação de contratos

4. Responder a mudanças mais que seguir um plano

Princípios

- Cliente tem prioridade com relação a metodologia
- Todos devem fazer parte do desenvolvimento, incluindo usuários, gerência, pedido, e outros
- As entregas devem ser feitas
  - De forma constante
  - Por prioridade que o cliente definir
  - Mantendo a simplicidade do sistema

Scrum

- No jogo de Rugby, o Scrum corresponde a uma reunião breve feita antes dos jogadores iniciarem um lance

- A metodologia Scrum define 3 papéis
  - Dono do produto (Product Owner - PO)
  - Mestre do Scrum (Scrum Master)
  - Time de desenvolvimento (Development Team)

Product Owner (PO)

- Pessoa responsável pelo produto a ser entregue
- Determina
  - O que será desenvolvido
  - E em qual ordem
- Faz a comunicação entre
  - Time de desenvolvimento
  - Entidades e pessoas envolvidas
- Garante a correta visão do que está sendo desenvolvido

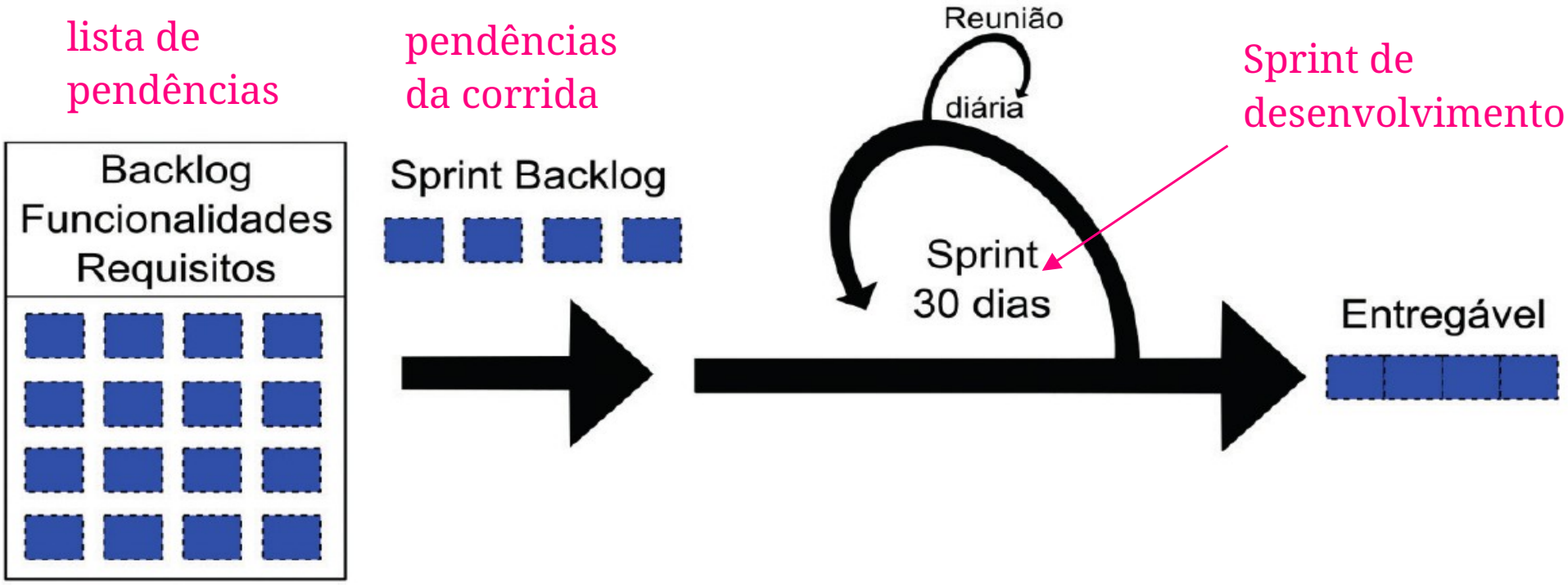
Scrum Master

- Possui familiaridade pela etapa atual de desenvolvimento podendo ajudar toda equipe
- Ajuda a manter os princípios, valores e papéis do Scrum
- Isola o time de desenvolvimento de interferências externas
- Resolve conflitos e problemas técnicos e pessoais para garantir a produtividade
- Não tem papel de gerência externa

Time de desenvolvimento

- Grupo de pessoas que se auto-organiza para atingir o objetivo proposto pelo PO
- Com 5 a 9 pessoas, coletivamente possui todas as habilidades necessárias para realizar o projeto

Figura 3.9 | Visão geral das etapas do *scrum*



- Backlog (lista de pendências geral)
  - PO define o conjunto de funcionalidades, junto com
    - Scrum Master
    - Equipe de desenvolvimento
  - A equipe de desenvolvimento pode opinar em relação aos requisitos, mas o PO tem prioridade no processo
- Sprint backlog (lista de pendências específica da corrida)
  - Conjunto de funcionalidades selecionadas pelo PO, com orientação técnica do
    - Scrum master
    - Time de desenvolvimento

- Corrida (Sprint)
  - As tarefas devem ser realizadas e entregues em certo período de tempo
- Ao final de cada *sprint* é gerado um entregável para o cliente
- Todos os dias são feitas reuniões com a equipe de desenvolvimento
- Backlog
  - Criado a partir de “histórias” de usuários trazidos pelo PO
  - 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>
- <tipo de usuário> explica qual usuário fará a ação
  - Ex.: usuário padrão, administrador, outros
- <objetivo> descreve o que deve ser feito, contendo
  - Suas necessidades
  - Justificativas para implementação

- Quem explica a razão
  - Explica as necessidades daquela funcionalidade
  - E em que momento deve ser priorizada
- Essas histórias são transformadas em tarefas
- O processo consiste em dividir as histórias em tarefas
  - Com nome (substantivo), e
  - Estimativa de tempo
- O tempo que se espera para concluir uma tarefa é utilizado para planejar o Sprint

- Essas tarefas levam em conta a criação das classes
  - Estas serão criadas a partir dos substantivos (nomes) das frases
- As atributos das classes poderão ser identificados pelos
  - Adjetivos (qualidades e estados)
  - Relacionamentos entre classes através dos verbos

Uma forma de organizar as tarefas

Figura 3.10 | *Scrum board* para controle das tarefas e histórias

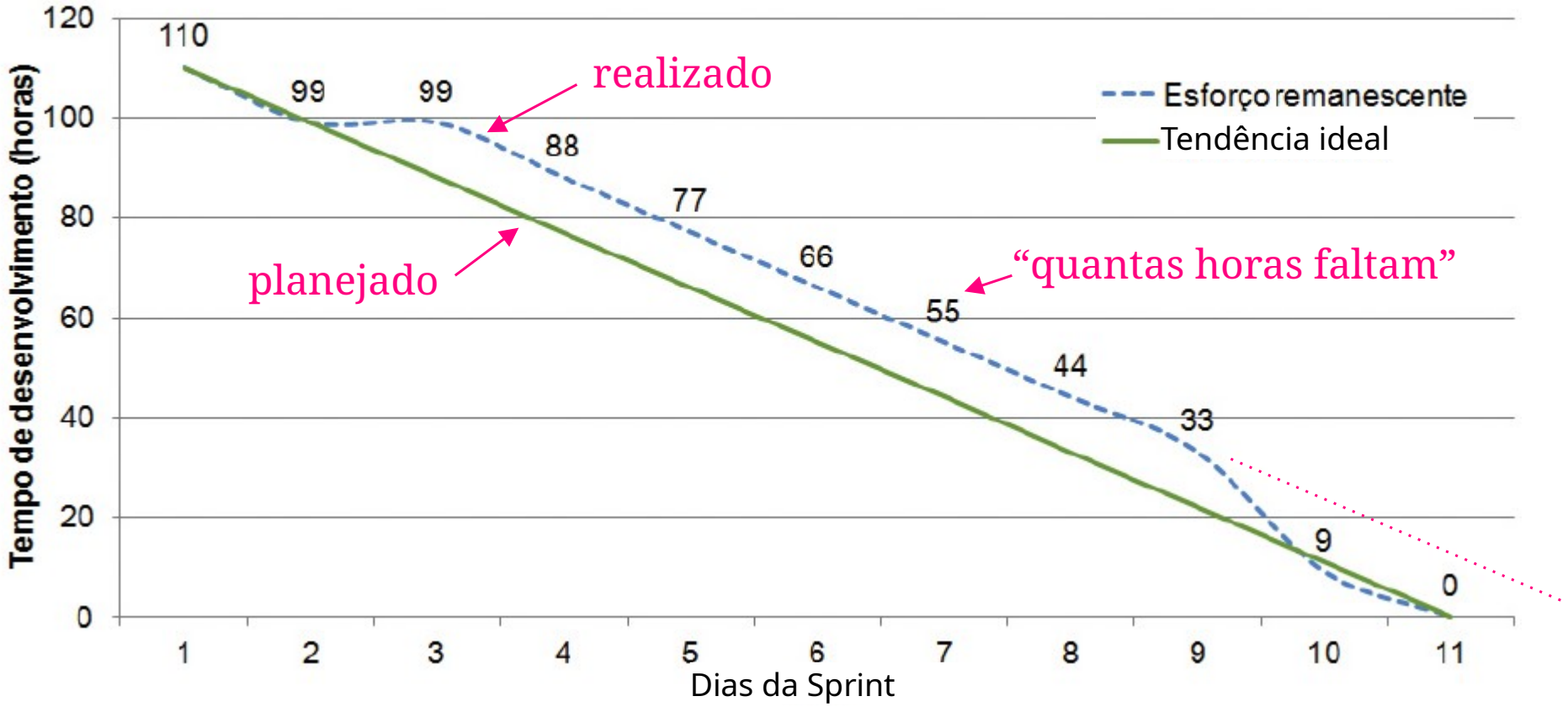
- Pode ser utilizado um quadro branco ou até uma parede com blocos de papel adesivo para organizar
  - Histórias de usuários
  - Tarefas iniciadas
  - O que está em progresso
  - O que já foi finalizado
- As tarefas devem ter
  - Conceitos de orientação a objetos (OO)
    - Devem ser fáceis de implementar usando OO
  - Formas corretas de planejamento
    - Ex.: usar uma metodologia

- A atualização do quadro com as histórias e as tarefas que estão sendo feitas pode ocorrer
  - No momento em que chega nova demanda, ou
  - Durante a Sprint de desenvolvimento (durante os 30 dias)
- Todos os dias, na reunião diária
  - As tarefas são atualizadas
  - Todos informam o que foi feito

Previsto x realizado (burndown)

Figura 3.11 | Exemplo do gráfico *burndown*

Sprint de 10 dias



- Além do quadro, é feito também um gráfico
- Mostra o T_{tot} necessário para o término da Sprint

- Nos primeiros dois dias é possível reparar que o desenvolvimento estava indo como o planejado
  - A linha tracejada coincide com a linha contínua
- No primeiro dia, a quantidade de horas planejadas (110) dividida pelo número de dias da Sprint (11):
  - 1º dia: 110 / 11 = 10 horas/dia
    - Está de acordo com o valor planejado informado pela equipe
- No dia 3 o tempo reportado foi mais baixo, deixando horas a serem realizadas nos outros dias
  - 3º dia: 99 / 8 = ~12 horas/dia para poder completar
- Quase ao final da Sprint foi possível realizar as tarefas

- Durante a Sprint, o comprometimento é sempre com a entrega do que o PO apresentou como prioridade
- Os desenvolvedores podem incluir no Backlog
  - Elementos de documentação
  - Testes
- Metodologias ágeis trouxeram fatores mais humanos para que o produto final seja realmente o que o cliente espera
- Tornam o desenvolvimento mais exato, reduzindo:
  - Retrabalhos
  - Atritos

- Métodos ágeis possuem aspectos
  - Incrementais
  - Humanos
  - Interativos
- O sistema em seu início tem uma estrutura orientada a objetos
  - Os métodos são bem planejados
    - Coesão (apenas uma única responsabilidade)
    - Acoplamento (dependências entre classes)

Refatoração

- Correções de problemas são sempre necessárias
- As correções tendem a alterar a estrutura do software
- Refatoração
  - Melhora um código que foi mal elaborado
  - Altera a estrutura de um código sem modificar o seu comportamento
  - Utiliza tempo no Sprint para alterar o código que já foi implementado e está funcionando

Razões para a refatoração

- Nomes de variáveis, métodos ou atributos sem significado
- Partes duplicadas do código
- Uso de bibliotecas ou de outros projetos
- Partes que não passaram nos testes
  - Testes unitários forçam a refatoração
- Tempo alto gasto em correções de
  - Bugs
  - Problemas em módulos de terceiros

- Momentos indicados para que seja feito o processo de correção do código
  - Quando se adiciona uma nova característica
    - É possível ver quais partes são utilizadas
      - E identificar eventuais problemas
  - No próprio processo de correção de bugs pode-se averiguar as condições do código e aplicar correções
  - Em casos mais complexos, pode ser necessário fazer uma revisão de código e aplicar a refatoração, para
    - Minimizar o tempo das correções
    - Melhorar o entendimento do código

- Refatoração é
  - Revisão e melhor entendimento de cada parte do código
  - Observar o código e avaliar se é necessário melhorar a base já implementada para que o projeto tenha uma duração longa
- Existem metodologias precisas para descrever como um código deve ser refeito para se tornar melhor

Metodologia

- 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 fazer ações específicas de maneira independente, buscando
  - Alta coesão (apenas uma única responsabilidade)
  - Baixo acoplamento (dependências entre classes)

- 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 projeto
  - Os testes precisam ser planejados