Universidade Estadual de Maringá
Departamento de Informática

Processos de software

Prof.ª Juliana Keiko Yamaguchi
março de 2019

Objetivos

- Definição de processo de software
- Modelos de Ciclo de Vida do software

Introdução

http://vidadeprogramador.com.br/?s=metodologia

Introdução

- **Processo:** Sequência de passos para realização de uma tarefa.
- Uma tarefa geralmente é formada por um conjunto de ações numa ordem definida
  - Antes de assar um bolo é preciso misturar os ingredientes
  - Usualmente, a argamassa é passada na parede de uma construção depois da instalação elétrica pronta

(Pfleeger, 1998)

Processo de software
Definição

- **Processo de software**
  - Conjunto de atividades para a produção de software (Sommerville, 2001)
- Quando um processo envolve a construção de um produto, este processo pode ser denominado por ciclo de vida
- Portanto, o processo de desenvolvimento do software também é conhecido por ciclo de vida do software (Pfleeger, 1998)

Processo de software
Por que são importantes?

- Usar processo de software garante a qualidade final do produto?
- Quais as vantagens de seguir um processo de software?

Processo de software
Por que são importantes?

- Vantagens na definição de um processo:
  - Facilidade de treinamento e redução de tempo (retrabalho);
  - Possibilidade de melhorias:
    - No processo: aprimoramento das atividades;
    - No produto: decorrente da melhoria do processo.

Modelos de ciclo de vida

- Os modelos de ciclo de vida do software são consequência:
  - da crise de software
  - e da necessidade de se ver o processo de desenvolvimento de software como uma engenharia.

Modelos de ciclo de vida

- Embora existam diferentes processos de software, existem atividades comum a todos os processos (Sommerville, 2001)
  - **Especificação:** definição das funcionalidades e restrições em suas operações
  - **Projeto e implementação:** construção do software de acordo com a especificação
  - **Validação:** para assegurar que o produto corresponde às necessidades do usuário
  - **Evolução:** amadurecimento do produto para novas necessidades do usuário

Ciclo de vida do software

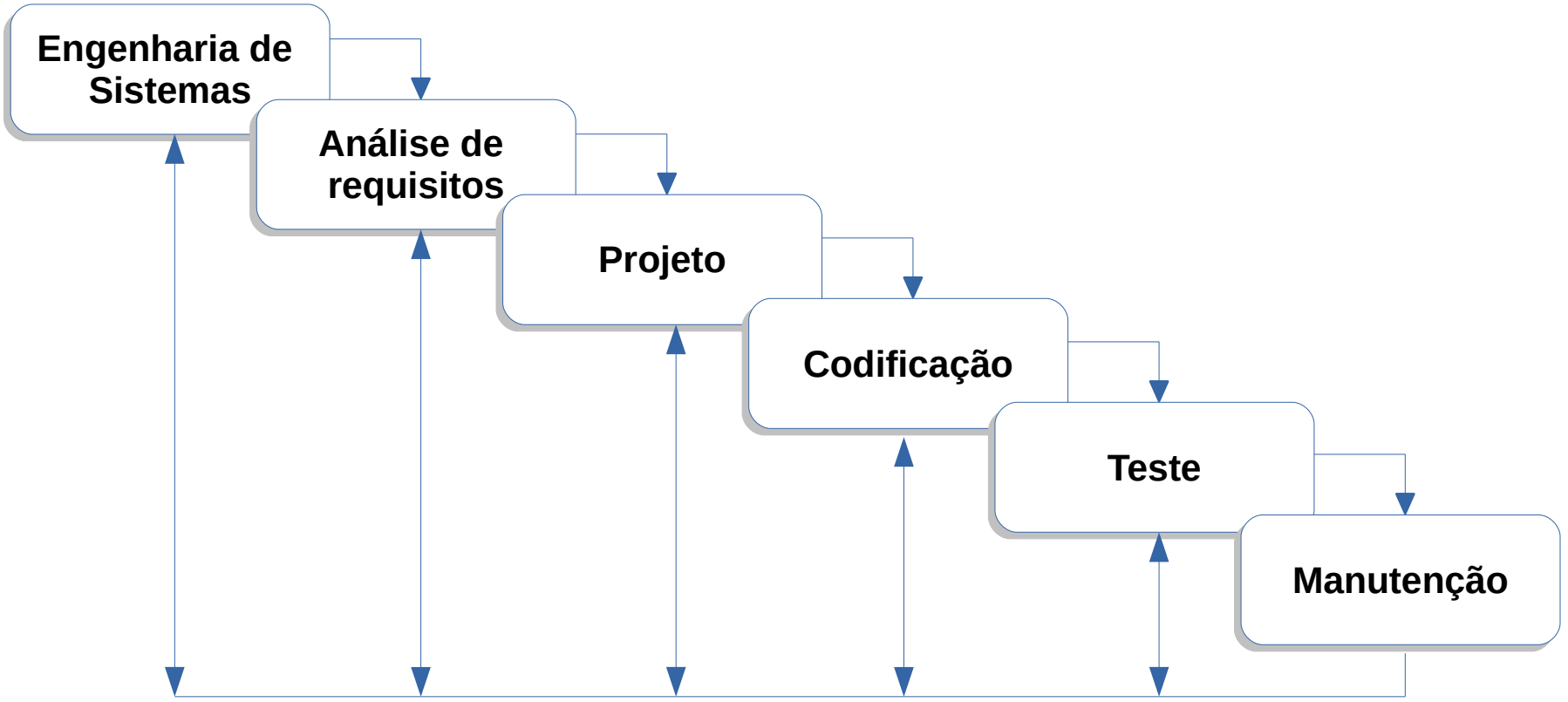
- **Atividades de proteção:** complementam as fases e passos descritos na visão genérica da Engenharia de Software
  - **Revisão:** garantir que a qualidade seja mantida à medida que cada etapa do ciclo é concluída.
  - **Documentação:** garantir que informações completas sobre o software estejam disponíveis para uso posterior.
  - **Controle de mudanças:** acompanhar as mudanças que possam ser aprovadas.

Modelos de ciclo de vida

- Cascata
- Prototipação
- Espiral
- Técnicas de quarta geração
- Processos ágeis
  - *Extreme Programming* (XP)
  - Scrum

Modelos de ciclo de vida

Modelo cascata (clássico)



Modelos de ciclo de vida
Modelo cascata (clássico)

- Engenharia de sistemas
  - o domínio da aplicação que deverá ser desenvolvido é conhecido pelo analista.
- Análise dos requisitos
  - os requisitos coletados são analisados.
- Projeto
  - realiza-se a descrição da arquitetura do sistema: estrutura de dados, arquitetura de software, detalhes procedimentais e caracterização da interface.

Modelos de ciclo de vida
Modelo cascata (clássico)

- Implementação
  - codificação dos programas pertencentes ao sistema.
- Testes
  - verificar e validar o software gerado.
- Implantação
  - o software é instalado no ambiente do cliente.
- Manutenção
  - as atividades focalizam-se na correção de erros, adaptação ambiental e incorporação de novas funções.

Modelos de ciclo de vida
Modelo cascata (clássico)

Vantagens	Desvantagens
▪ Subprocessos executados em estrita sequência. ▪ Permite controle bem definido, facilitando a gestão e planejamento.	▪ Projetos reais raramente seguem o fluxo sequencial proposto por este modelo: na maioria dos casos existe interação e superposição. ▪ Raramente os clientes (usuários) declaram todas as exigências de uma vez, no início do projeto. ▪ Uma versão de trabalho dos programas não estará disponível até um ponto adiantado no cronograma do projeto: é geralmente difícil convencer o usuário de que é preciso paciência.

Modelos de ciclo de vida
Prototipação

- Apropriado para quando o cliente definiu um conjunto de objetivos gerais para o software, mas não identificou requisitos de entrada, processamento e saída com detalhes.
- Possibilita ao desenvolvedor criar um modelo do software que deve ser construído.
- Idealmente, o modelo (protótipo) serve como um mecanismo para identificar os requisitos de software.

Modelos de ciclo de vida
Prototipação

- A primeira fase prevê o desenvolvimento de um programa para o usuário experimentar, com o objetivo de estabelecer os requisitos do sistema.
- Para sistemas grandes e complicados.
- Quando não existe um sistema anterior ou um sistema manual que ajude a especificar os requerimentos.
- O software deve ser reimplementado na fase seguinte.

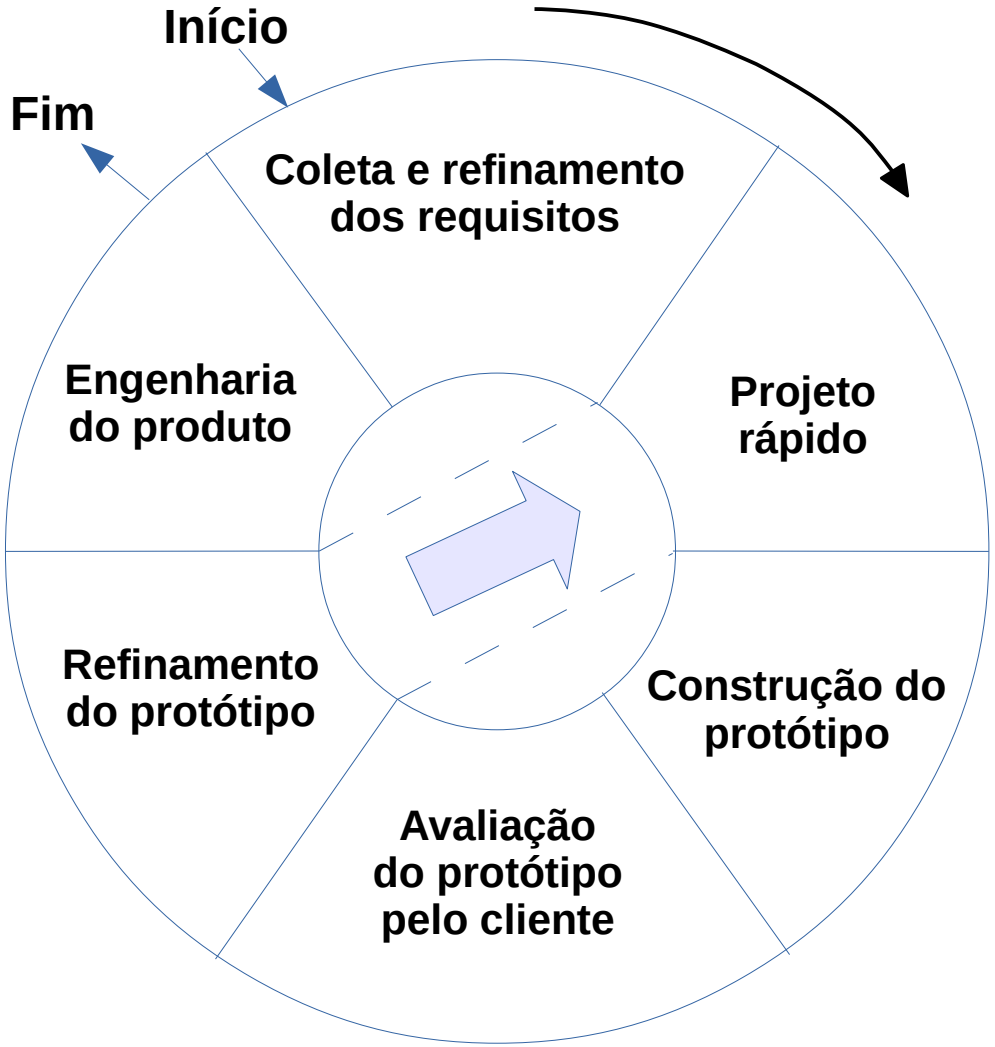
Modelos de ciclo de vida
Prototipação

- Os objetivos do protótipo devem estar bem claros antes do início da codificação.
- Possíveis objetivos:
  - Entender os requerimentos dos usuários
  - Definir a interface com os usuários
  - Demonstrar a viabilidade do sistema para os gerentes
  - Deve estar claramente acordado entre cliente e desenvolvedor que o protótipo será descartado

Modelos de ciclo de vida

http://vidadeprogramador.com.br/?s=prot%C3%B3tipo

Modelos de ciclo de vida
Prototipação



Modelos de ciclo de vida
Prototipação

1. Coleta e refinamento dos requisitos: desenvolvedor e cliente definem os objetivos gerais do software, identificam quais requisitos são conhecidos e as áreas que necessitam de definições adicionais.

2. Projeto rápido: representação dos aspectos do software que são visíveis ao usuário (abordagens de entrada e formatos de saída)

3. Construção do protótipo: implementação do projeto rápido

Modelos de ciclo de vida
Prototipação

4. Avaliação do protótipo: cliente e desenvolvedor avaliam o protótipo

5. Refinamento do protótipo: cliente e desenvolvedor refinam os requisitos do software a ser desenvolvido.

- Ocorre neste ponto um processo de iteração que pode conduzir a atividade 1 até que as necessidades do cliente sejam satisfeitas e o desenvolvedor compreenda o que precisa ser feito.

Modelos de ciclo de vida
Prototipação

6. Engenharia do produto: identificados os requisitos, o protótipo deve ser descartado e a versão de produção deve ser construída considerando os critérios de qualidade.

Modelos de ciclo de vida
Prototipação

Vantagens	Desvantagens
▪ Protótipos contribuem para melhorar a qualidade da especificação dos futuros programas, o que leva à diminuição dos gastos com manutenção. ▪ Em alguns casos, o treinamento dos usuários pode até ser feito antes do produto ficar pronto. ▪ Algumas partes do protótipo podem vir a ser usadas no desenvolvimento do sistema final.	▪ Em geral o grande argumento contra a construção de protótipos é o custo. Construir um protótipo pode não ser tão rápido do que construir o sistema final, acarretando em atrasos. ▪ O cliente vê algo que parece ser uma versão do software desejado e não entende porque o produto precisa ser reconstruído. ▪ Muitas das concessões feitas na implementação do protótipo visando a construção rápida podem vir a fazer parte do sistema final, comprometendo a qualidade do produto.

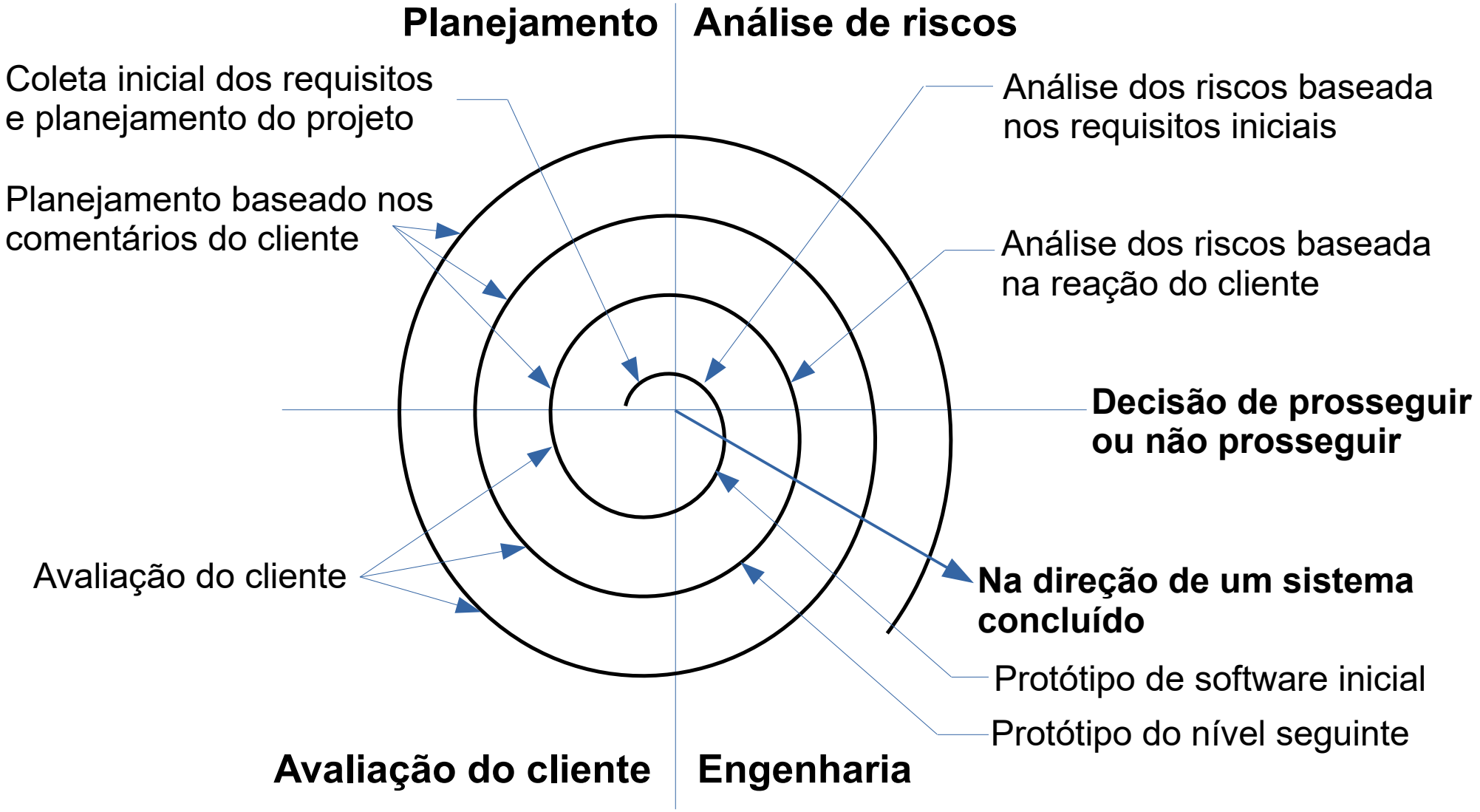
Modelos de ciclo de vida
Espiral

- O modelo espiral foi criado visando abranger as melhores características do modelo clássico e da prototipagem (Boehm, 1988)
- A cada ciclo da espiral, versões progressivamente mais completas do software são construídas.

Modelos de ciclo de vida
Espiral

- Acrescenta aspectos gerenciais ao processo de desenvolvimento de software
  - análise de riscos em intervalos regulares do processo de desenvolvimento de software;
  - planejamento;
  - controle;
  - tomada de decisão.

Modelos de ciclo de vida
Espiral



Modelos de ciclo de vida
Espiral

1. Planejamento: determinação dos objetivos, alternativas e restrições

2. Análise de risco: análise das alternativas e identificação / resolução dos riscos

3. Engenharia: desenvolvimento do produto no nível seguinte

4. Avaliação do cliente: avaliação do produto e planejamento das novas fases

Modelos de ciclo de vida
Espiral

Vantagens	Desvantagens
▪ Modelo evolutivo possibilita uma maior integração entre as fases e facilita a depuração e a manutenção do sistema. ▪ Permite que o projetista e cliente possam entender e reagir aos riscos em cada etapa evolutiva.	▪ Avaliação dos riscos exige muita experiência – gerente de projeto é responsável por minimizar os riscos. ▪ Risco do projeto está relacionado a incerteza quanto a aspectos operacionais (controle de informação), organizacionais (processo de construção do produto e papéis do indivíduo) e contratuais (fornecedores).

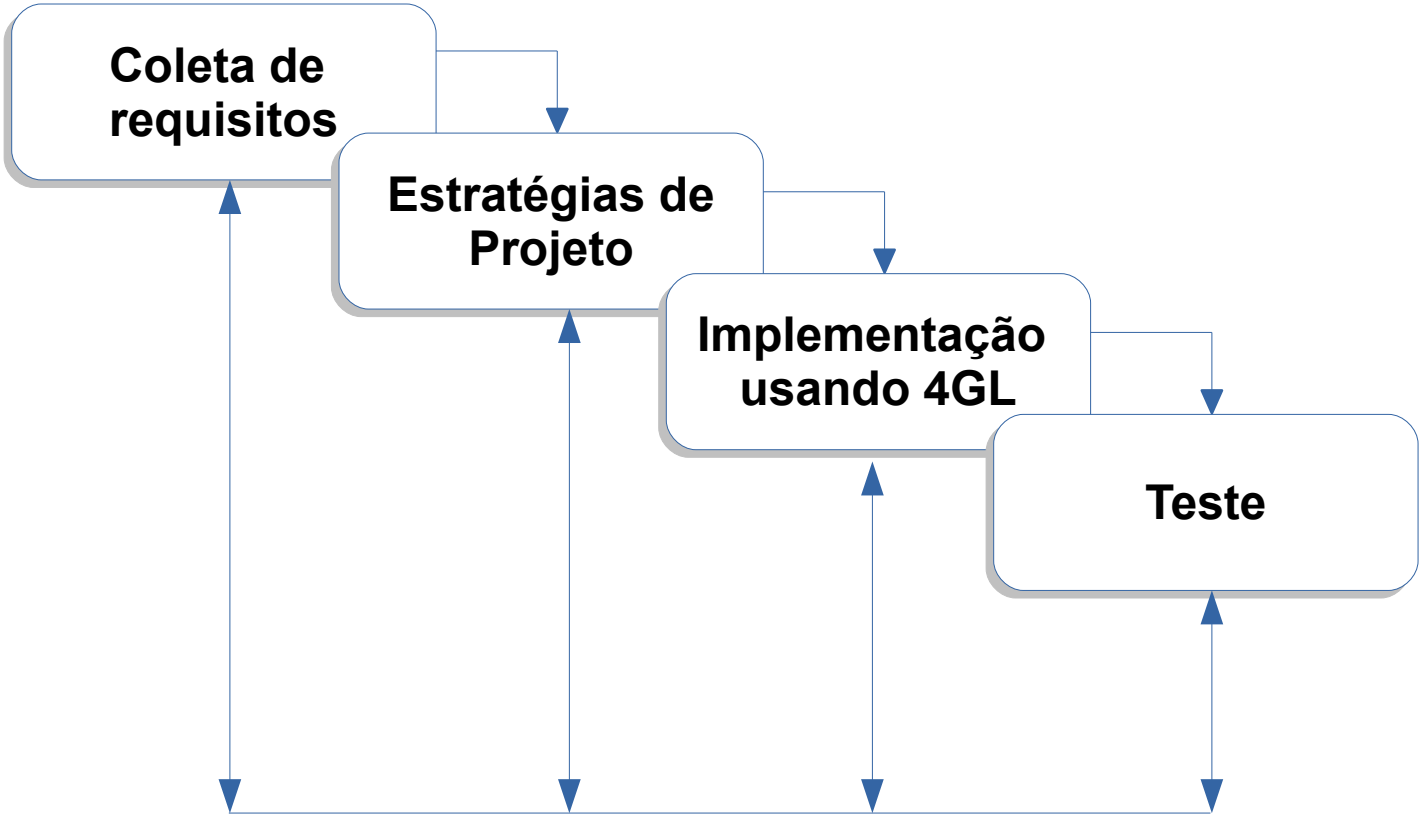
Modelos de ciclo de vida
Técnicas de quarta geração (4GT)

- Abrange um amplo conjunto de ferramentas de software que permitem ao desenvolvedor especificar alguma característica do software em um nível elevado
- A partir desta especificação, um código executável é gerado.
- É um paradigma de desenvolvimento e não um modelo propriamente dito

Modelos de ciclo de vida
Técnicas de quarta geração (4GT)

- Ferramentas:
  - Linguagens não procedimentais para:
    - acesso a Bancos de Dados (SQL)
    - geração de relatórios (XML)
    - definição e manipulação de telas (Delphi, Visual Basic)
  - Capacidade gráfica de alto nível.
  - Geração de código
    - geração automática de HTML e linguagens similares para criação de páginas Web

Modelos de ciclo de vida
Técnicas de quarta geração (4GT)



Modelos de ciclo de vida
Técnicas de quarta geração (4GT)

1. Coleta dos requisitos: o cliente descreve os requisitos os quais são traduzidos para um protótipo operacional
   - o cliente pode ser incapaz de especificar as informações de um modo que uma ferramenta 4GL possa consumir

Modelos de ciclo de vida
Técnicas de quarta geração (4GT)

2. Estratégia de projeto: para pequenas aplicações é possível mover-se do passo da coleta dos requisitos para o passo de implementação usando uma linguagem de quarta geração.

- Para grandes projetos é necessário desenvolver uma estratégia de projeto.
- De outro modo ocorrerão os mesmos problemas encontrados quando se usa abordagem convencional (baixa qualidade).

Modelos de ciclo de vida
Técnicas de quarta geração (4GT)

3. Implementação usando 4GL: os resultados desejados são representados de modo que haja geração automática de código. Deve existir uma estrutura de dados com informações relevantes e que seja acessível pela 4GL.

4. Teste: o desenvolvedor deve efetuar testes e desenvolver uma documentação significativa. O software desenvolvido deve ser construído de maneira que a manutenção possa ser efetuada prontamente.

Modelos de ciclo de vida
Técnicas de quarta geração (4GT)

Vantagens	Desvantagens
▪ Redução no tempo de desenvolvimento de software em função de uma maior produtividade.	▪ Ferramentas nem sempre são tão mais fáceis de usar do que ferramentas mais convencionais ▪ O código gerado pode ser ineficiente.

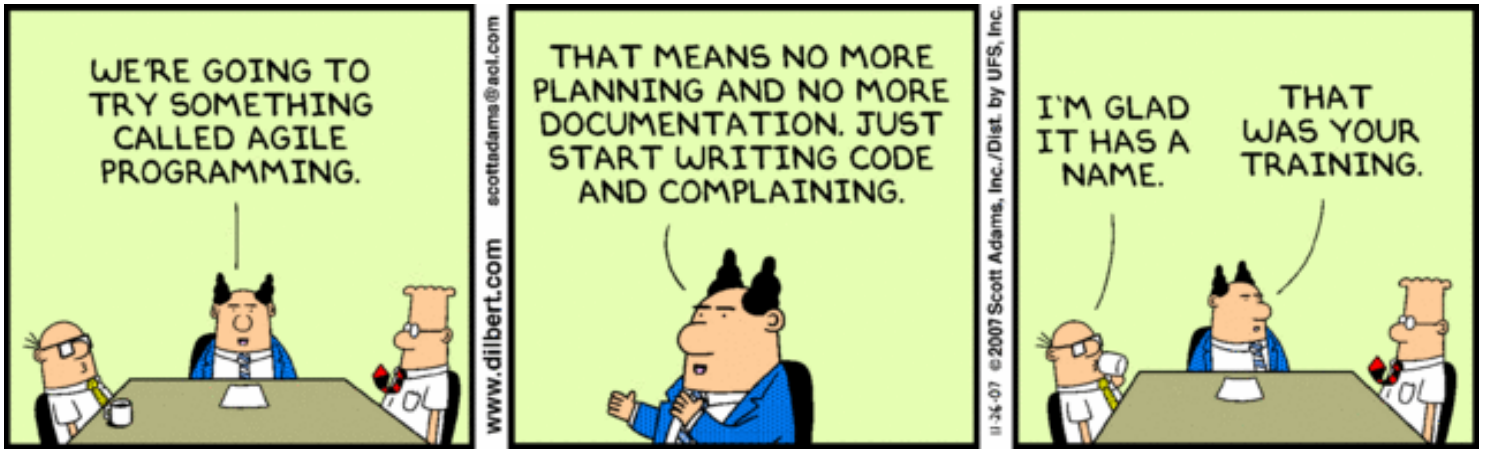
Processos ágeis

- Processos ágeis surgiram como alternativa ao processos tradicionais
- Desburocratização do processo de desenvolvimento
- Manifesto ágil: http://www.agilemanifesto.org/
  - documento que reúne os princípios e práticas desta metodologia de desenvolvimento. Mais tarde, algumas pessoas formaram a *Agile Alliance*, uma organização não lucrativa que promove o desenvolvimento ágil.

Processos ágeis

- Adaptação e fortalecimento com as mudanças
- Versões de software executável em curto espaço de tempo
- São priorizadas as funcionalidades que agregam maior valor ao negócio
- Não “documentar por documentar”
  - somente quando for estritamente necessário
  - documentação sobre pontos-chave da arquitetura, descrições do funcionamento do sistema

Processos ágeis



Processos ágeis
Princípios do desenvolvimento ágil

- Garantir a satisfação do consumidor entregando rápido e continuamente softwares funcionais.
- Até mesmo mudanças tardias de escopo no projeto são bem-vindas.
- Softwares funcionais são entregues frequentemente (semanas, ao invés de meses).
- Cooperação constante entre pessoas que entendem do 'negócio' e desenvolvedores.

Processos ágeis
Princípios do desenvolvimento ágil

- Projetos surgem através de indivíduos motivados. Deve existir uma relação de confiança recíproca.
- Indivíduos e interações mais do que processos e ferramentas: comunicação direta (conversação) entre os membros da equipe.
- Software funcionando é a medida primária de progresso.
- Colaboração com clientes mais do que negociação de contratos: manter ritmo sustentável de desenvolvimento.

Processos ágeis
Princípios do desenvolvimento ágil

- Atenção contínua à excelência técnica e bom projeto aumentam a agilidade.
- Simplicidade é essencial.
- As melhores arquiteturas, levantamentos de requisitos e projetos emergem de equipes auto-organizáveis.
- Em intervalos regulares, a equipe reflete sobre como tornar-se mais eficaz e incorpora atitudes com esta finalidade.

Modelos ágeis

- Extreme Programming (XP)
- Scrum

Extreme Programming (XP)

- Criado em 1999 por Kent Beck e Ward Cunningham, quando do lançamento do livro *Extreme Programming Explained*
- Voltado ao desenvolvimento de produtos com requisitos vagos e em constante mudança
- Baseado em valores, princípios e práticas

Extreme Programming (XP)
Valores

- Comunicação: prioriza a comunicação pessoal e oral ao invés da escrita
- Simplicidade: XP recomenda manter o sistema simples, de forma a não gerar mais trabalho desnecessário
- *Feedback:* ao longo do desenvolvimento, é muito importante que exista *feedback* dentro do time de desenvolvimento e com o cliente e/ou demais envolvidos

Extreme Programming (XP)
Valores

- Coragem: o time pode ter coragem de refatorar, pois sabe que os testes irão detectar erros imediatamente, o cliente pode decidir com mais coragem, quando avalia o software funcional após cada versão, sabendo que pode priorizar as funcionalidades que lhe são mais importantes
- Respeito: valorizar a relação entre membros da equipe e, também, a de cada membro com o projeto e sua instituição

Extreme Programming (XP)
Princípios básicos

- *Feedback* rápido
- Presumir simplicidade
- Mudanças incrementais
- Abraçar mudanças
- Trabalho de alta qualidade

Extreme Programming (XP)
Práticas

- Cliente disponível ou presente: XP sugere que o cliente esteja no dia-a-dia do projeto, acompanhando os passos dos desenvolvedores, onde a sua ausência representa sérios riscos ao projeto.
- Planejamento: XP utiliza o planejamento para assegurar que a equipe de desenvolvimento esteja trabalhando naquilo que gere o máximo de valor para o cliente. Este planejamento é feito várias vezes durante o projeto.

Extreme Programming (XP)
Práticas

- *Stand up meeting*: o dia de trabalho de uma equipe XP sempre começa com um *stand up meeting*. É uma reunião de aproximadamente 15 minutos de duração e onde seus integrantes permaneçam preferencialmente em pé.
  - reunião em pé é mais rápida, evita bate-papos paralelos e faz os integrantes irem diretamente ao assunto.
  - A reunião tem por objetivo atualizar a equipe sobre o que foi implementado no dia anterior e trocar experiências das dificuldades enfrentadas.

Extreme Programming (XP)
Práticas

- Programação em par (*pair-programming*): XP exige que todo e qualquer código implementado no projeto seja efetuado em dupla.
  - Dois desenvolvedores trabalham no mesmo problema, ao mesmo tempo e no mesmo computador.
  - Um deles é responsável pela digitação (condutor) e outro acompanhando o trabalho do parceiro (navegador).

Extreme Programming (XP)

http://vidadeprogramador.com.br/2012/05/22/pair-programming/

Extreme Programming (XP)
Práticas

- *Refactoring:* XP prega que todo desenvolvedor ao encontrar um código duplicado, pouco legível, mal codificado, sem padronização, lento, com código legado ou uso incorreto de outras implementações, tem por obrigação alterar este código deixando-o mais legível e simples, porém esta alteração não pode mudar o comportamento do código em questão
  - Código coletivo: cada desenvolvedor tem acesso a qualquer parte do sistema e tem liberdade para alterá-la a qualquer momento e sem qualquer tipo de aviso.

Extreme Programming (XP)
Práticas

- Desenvolvimento guiado por testes: Todo código implementando deve coexistir com um teste automatizado, assim garantindo o pleno funcionamento da nova funcionalidade.
  - é com base nestes testes automatizados que qualquer desenvolvedor terá coragem para alterar uma parte do código que não tenha sido implementada por ele, já que executando os testes automatizados poderá verificar instantaneamente se a modificação efetuada alterou o comportamento do sistema.

Extreme Programming (XP)
Ciclo de vida

- Planejamento
- Projeto
- Codificação
- Teste

Extreme Programming (XP)
Ciclo de vida – Planejamento

- Conjunto de histórias formuladas com o cliente que descrevem características e funcionalidades necessárias para o software a ser construído
- Cada história representa uma atividade no controle do processo e o cliente atribui-lhe um valor de prioridade
- Os membros da equipe analisam esta lista e atribui-lhe custos

Extreme Programming (XP)
Ciclo de vida – Planejamento

- Se a história precisar de mais de três semanas, pede-se ao cliente que a divida
- Novas histórias podem ser adicionadas a qualquer momento.
- O próximo passo é a equipe, em colaboração com o cliente, decidir que histórias vão ficar prontas na próxima iteração e em que data.

Extreme Programming (XP)
Ciclo de vida – Planejamento

Extreme Programming (XP)
Ciclo de vida – Projeto

- Filosofia KIS (*keep it simple*)
  - é desencorajado o desenvolvimento de uma funcionalidade extra só porque o desenvolvedor acha que mais tarde pode precisar.
- XP encoraja a refatoração

Extreme Programming (XP)
Ciclo de vida – Codificação

- Antes do código, recomenda-se que se crie uma bateria de testes unitários para que a história fique satisfeita.
- Então o foco dos programadores é a satisfação destes testes unitários (programação em dupla)

Extreme Programming (XP)
Ciclo de vida – Teste

- Os testes unitários, são mantidos ao longo das várias iterações e passam a fazer parte de uma bateria de testes de regressão, que não é mais do que todos os testes unitários agrupados para serem testados periodicamente de uma vez em períodos curtos (pode ser de horas, ao final do dia, final da semana).
- A ideia é confirmar que nada deixou de funcionar.

Scrum

- Scrum é fundamentado na teoria de controle de processos empíricos.
- Emprega uma abordagem iterativa e incremental para otimizar a previsibilidade e controlar riscos.
- Três pilares sustentam cada implementação de controle de processos empíricos.
  - Transparência
  - Inspeção
  - Adaptação

Scrum
Transparência

- Garante que aspectos do processo que afetam o resultado devem ser visíveis para aqueles que gerenciam os resultados.
- Quando alguém que inspeciona um processo acredita que algo está pronto, isso deve ser equivalente à sua definição de pronto.

Scrum
Inspeção

- Os diversos aspectos do processo devem ser inspecionados com uma frequência suficiente para que variações inaceitáveis no processo possam ser detectadas.
- A frequência da inspeção deve levar em consideração que qualquer processo é modificado pelo próprio ato da inspeção.

Scrum
Adaptação

- Se o inspetor determinar, a partir da inspeção, que um ou mais aspectos do processo estão fora dos limites aceitáveis e que o produto resultante será inaceitável, ele deverá ajustar o processo ou o material sendo processado.
- Esse ajuste deve ser feito o mais rápido possível para minimizar desvios posteriores.

Scrum
Papéis

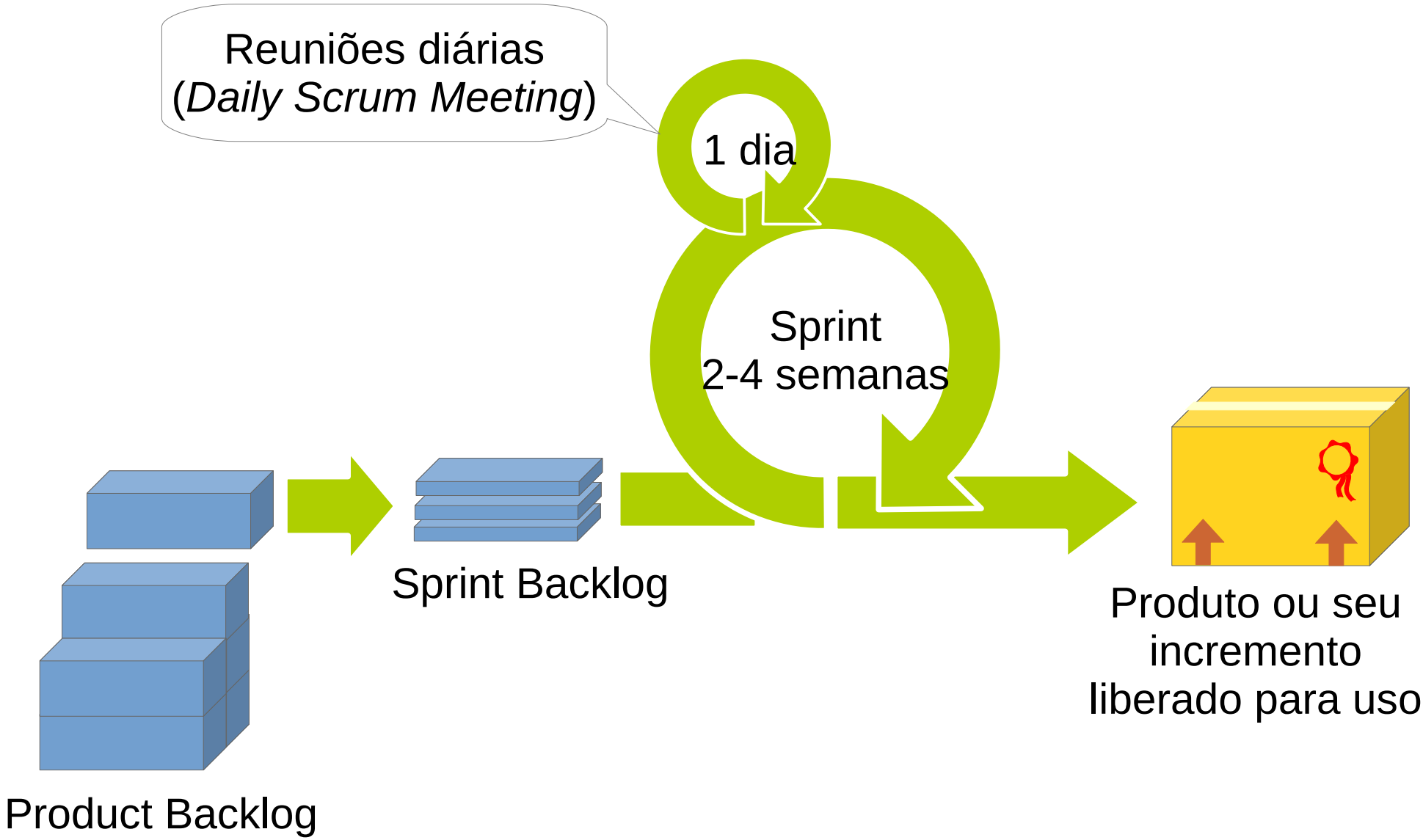
- Scrum Master
- Product Owner
- Time Scrum

http://www.implementingscrum.com/

Scrum
Papéis

- Scrum Master: responsável por garantir que o processo seja compreendido e seguido
- Product Owner: responsável por maximizar o valor do trabalho que o Time de Scrum faz
- Time: executa o trabalho propriamente dito
  - O Time consiste em desenvolvedores com todas as habilidades necessárias para transformar os requisitos do Product Owner em um pedaço potencialmente entregável do produto ao final da Sprint.

Scrum
Ciclo de vida



*Backlog – acúmulo de trabalho por fazer

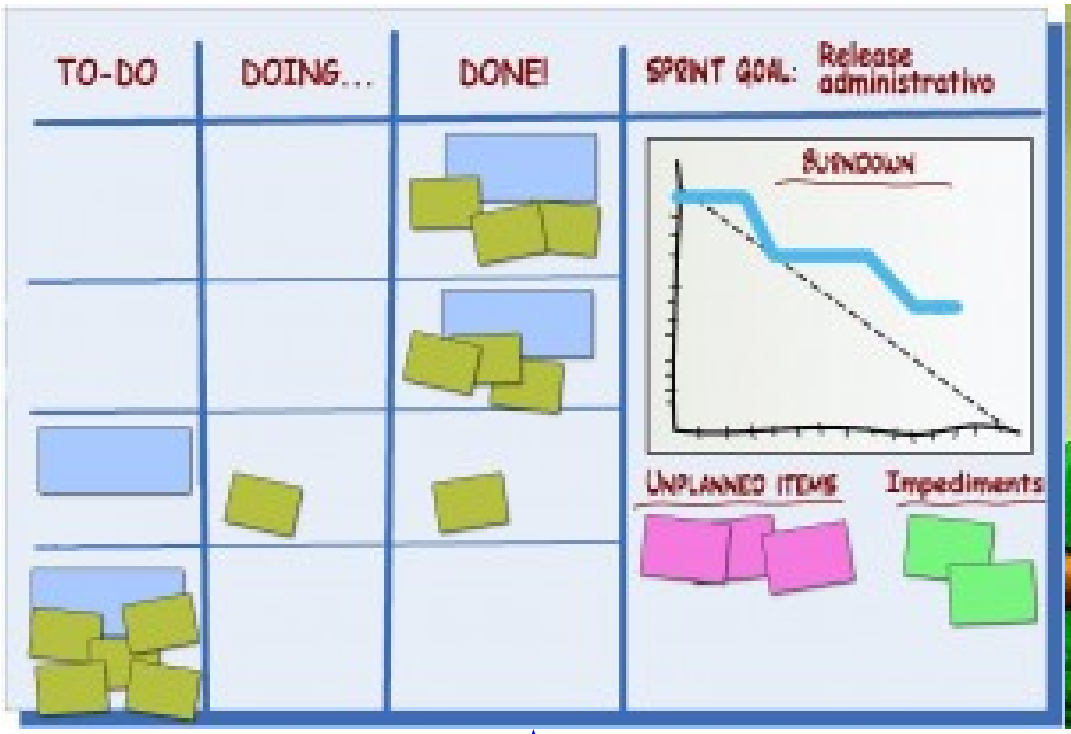
Scrum
Artefatos

- Product Backlog é uma lista priorizada de tudo que pode ser necessário no produto
- Sprint Backlog é uma lista de tarefas para transformar o Product Backlog em um incremento do produto potencialmente entregável por meio de uma Sprint

Scrum
Artefatos

- Burndown é uma medida do backlog restante pelo tempo
  - Burndown de Release: mede o Product Backlog restante ao longo do tempo de um plano de release.
  - Burndown de Sprint: mede os itens do Sprint Backlog restantes ao longo do tempo de uma Sprint

Scrum
Artefatos – Gráfico Burndown



Quadro Kanban

https://www.guiadoexcel.com.br/grafico-burndown-scrum-excel/

Scrum
Ciclo de vida

- O coração do Scrum é a **Sprint**, que é uma iteração de um mês ou menos, de duração consistente com o esforço de desenvolvimento
- Todas as Sprints têm como resultado um incremento do produto final que é potencialmente entregável.
- Cada Sprint começa imediatamente após a anterior.

Scrum

Reunião de planejamento da *Sprint*

- Para Sprint de 1 mês: 8 horas
- Para Sprint de 2 semanas: 4 horas
- Definição de “o quê” e “como” será feito do Product Backlog na próxima Sprint
  - Cabe somente ao Time a decisão de quanto do Backlog ele deve selecionar
  - Somente o Time pode avaliar o que ele é capaz de realizar na próxima Sprint

Scrum

Reunião de planejamento da *Sprint*

- Time define como transformará o Product Backlog selecionado durante a Reunião de Planejamento (o quê) em um incremento pronto.
- As tarefas devem ser decompostas para que possam ser feitas em menos de um dia. Essa lista de tarefas é chamada de Sprint Backlog.
- O time se auto-organiza para se responsabilizar pelo trabalho contido no Sprint Backlog.

Scrum
Revisão da *Sprint*

- Feita após a finalização de uma Sprint
- Durante a Revisão da Sprint, o Time de Scrum e as partes interessadas discutem sobre o que acabou de ser feito:
  - O Product Owner identifica o que foi feito e o que não foi feito
  - O Time discute sobre o que correu bem durante a Sprint e quais problemas foram enfrentados, além de como esses problemas foram resolvidos
  - O Time então demonstra o trabalho que está pronto e responde a questionamentos

Scrum
Retrospectiva da *Sprint*

- Após a Revisão da Sprint e antes da próxima reunião de Planejamento da Sprint, o Time de Scrum tem a reunião de Retrospectiva da Sprint
  - Esta é uma reunião com duração fixa de três horas para Sprints de um mês
- Nessa reunião, o Scrum Master encoraja o Time a revisar, dentro do modelo de trabalho e das práticas do processo do Scrum, seu processo de desenvolvimento, de forma a torná-lo mais eficaz e gratificante para a próxima Sprint.

XP versus Scrum

Extreme Programming	Scrum
Enfatiza um conjunto de regras e práticas de desenvolvimento.	Enfatiza o uso de um conjunto de padrões de processos de software.
Iterações demoram cerca de uma a duas semanas.	Iterações (sprints) demoram de duas semanas a um mês.

XP versus Scrum

Extreme Programming	Scrum
O trabalho definido em uma iteração que ainda não iniciado pode ser trocado por outra de maior prioridade.	O trabalho definido em uma sprint é estável, isto é, alterações não são introduzidas.
O time trabalha sobre as prioridades dos requisitos ditadas pelo cliente.	O Product Owner organiza as prioridades no Backlog e o time planeja a próxima iteração (sprint)

Considerações finais

- Não há um “processo perfeito”.
- Cada organização deve ajustar sua metodologia de trabalho combinando conceitos e práticas de diferentes processos conforme suas necessidades específicas.
- XP e Scrum se complementam.
- Existe o melhor modelo?
  - https://cio.com.br/usar-scrum-para-tudo-evite-essa-tentacao/
  - Acessado em março de 2019.

Considerações finais

- Ferramentas para gerenciar atividades do Scrum estão disponíveis na internet.
  - Exemplo:
  - https://www.siteware.com.br/projetos/ferramentas-para-scrum/
  - Acessado em março de 2019.
-

Referências

- Ramos, Ricardo Argenton. Processos de desenvolvimento de software. Notas de aula. UNIVASF. Disponível em:
  - http://www.univasf.edu.br/~ricardo.aramos/disciplinas/ESI2009_2/Aula02.pdf

    Acessado em março de 2015.
- Extreme Programming:
  - http://www.extremeprogramming.org/>
- Scrum:
  - http://www.scrum.org/
  - http://www.scrumalliance.org/pages/what_is_scrum

Referências

- PRESSMAN, R. S. Engenharia de Software: Uma abordagem profissional – McGrawHill Bookman, Porto Alegre, 2011, 7.ed.
- REZENDE, D. A. Engenharia de Software e Sistema de Informação, Brasport, 2005, 3ª ed., Rio de Janeiro.
- SOMMERVILLE, I. Engenharia de Software, Adisson Wesley, 2003, 6ª.ed., São Paulo.
- TOMÁS, M. R. S. Métodos ágeis: características, pontos fortes e fracos e possibilidades de aplicação. Relatório técnico. Universidade de Nova Lisboa, Portugal, 2009.