Engenharia de Software

Processos de Software

Profª Raquel Mini

raquelmini@ufmg.br

DEE / UFMG

Unidade II

- Processos de software
  - Ciclo de vida de software
  - Modelos de processo de software
  - Métodos ágeis de desenvolvimento de software
    - Manifesto ágil
    - Programação extrema (XP)
    - Scrum
  - RUP

Ian Sommerville. Engenharia de Software, 9ª Edição.
Pearson Education, 2011. Capítulo 2 e 3.

Ciclos de vida

- Conjunto da história de um software, desde a percepção de sua necessidade até sua retirada de operação
- Determina o caráter temporal do software
  - Começa antes do projeto de desenvolvimento
  - Termina após o esgotamento
- Surgiram na década de 70
  - Precursores dos atuais processos de software

Ciclos de vida

- Ciclos de vida são mais abrangentes:
  - Que a codificação
  - Que projetos de software

<table>
<tr><td rowspan="9">Ciclo de Vida</td><td colspan="4">Percepção da Necessidade</td></tr>
<tr><td rowspan="7">Desenvolvimento</td><td colspan="3">Concepção</td></tr>
<tr><td colspan="3">Elaboração</td></tr>
<tr><td rowspan="4">Construção</td><td colspan="2">Desenho arquitetônico</td></tr>
<tr><td rowspan="3">Liberação</td><td>Desenho detalhado</td></tr>
<tr><td>Codificação</td></tr>
<tr><td>Testes de unidade</td></tr>
<tr><td colspan="2">Testes de Aceitação</td></tr>
<tr><td colspan="3">Transição</td></tr>
<tr><td colspan="4">Operação</td></tr>
<tr><td colspan="4">Retirada</td></tr>
</table>

Processo

- É uma sequência de passos realizados com um propósito
- Integra pessoas, ferramentas e procedimentos
- É o que pessoas fazem, utilizando procedimentos, métodos, ferramentas, e equipamentos, para transformar matéria prima (*inputs*) em um produto (*outputs*) de valor para os seus clientes

Processo de software

- Conjunto de atividades relacionadas que levam à produção de um produto de software
- Um processo define
  - Quem faz, o que faz e quando fazer
  - Nem sempre diz como fazer
- Não existe um processo ideal
  - Organizações desenvolvem seus próprios processos
- Ciclo de vida corresponde ao aspecto temporal do processo de software

Processo de software

- Existem quatro atividades fundamentais comuns a todos os processos de software:
  - Especificação de software
  - Projeto e implementação de software
  - Validação de software
  - Evolução do software
- Existem também as atividades que dão apoio ao processo, como desenvolvimento e gerenciamento de configuração de software

Processo de software

- As descrições dos processos também incluem:
  - Produtos: resultados de uma atividade do processo
  - Papéis: refletem as responsabilidades das pessoas envolvidas no processo
  - Pré e pós-condições: declarações que são verdadeiras antes e depois de uma atividade do processo ser executada, ou um produto produzido

Processo de software

- Os processos são categorizados como:
  - **Dirigidos a planos**: todas as atividades são planejadas com antecedência e o progresso é avaliado por comparação com o planejamento inicial
  - **Processos ágeis**: o planejamento é gradativo e é mais fácil alterar o processo de maneira a refletir as necessidades de mudança dos clientes

Modelo de processo

- Representação simplificada de um processo de software
- Cada modelo representa uma perspectiva particular de um processo e, portanto, fornece informações parciais sobre ele
- Os modelos de processos não são mutuamente exclusivos
  - Organizações tendem a combinar partes de diferentes modelos em seus processos

Exemplos de modelos de processo

- Modelos de processo mais gerais (paradigmas de processo)
  - Modelo cascata
  - Desenvolvimento incremental
  - Engenharia de software orientada a reuso
- Modelos que lidam com mudanças
  - Prototipagem
  - Entrega incremental
  - Modelo espiral

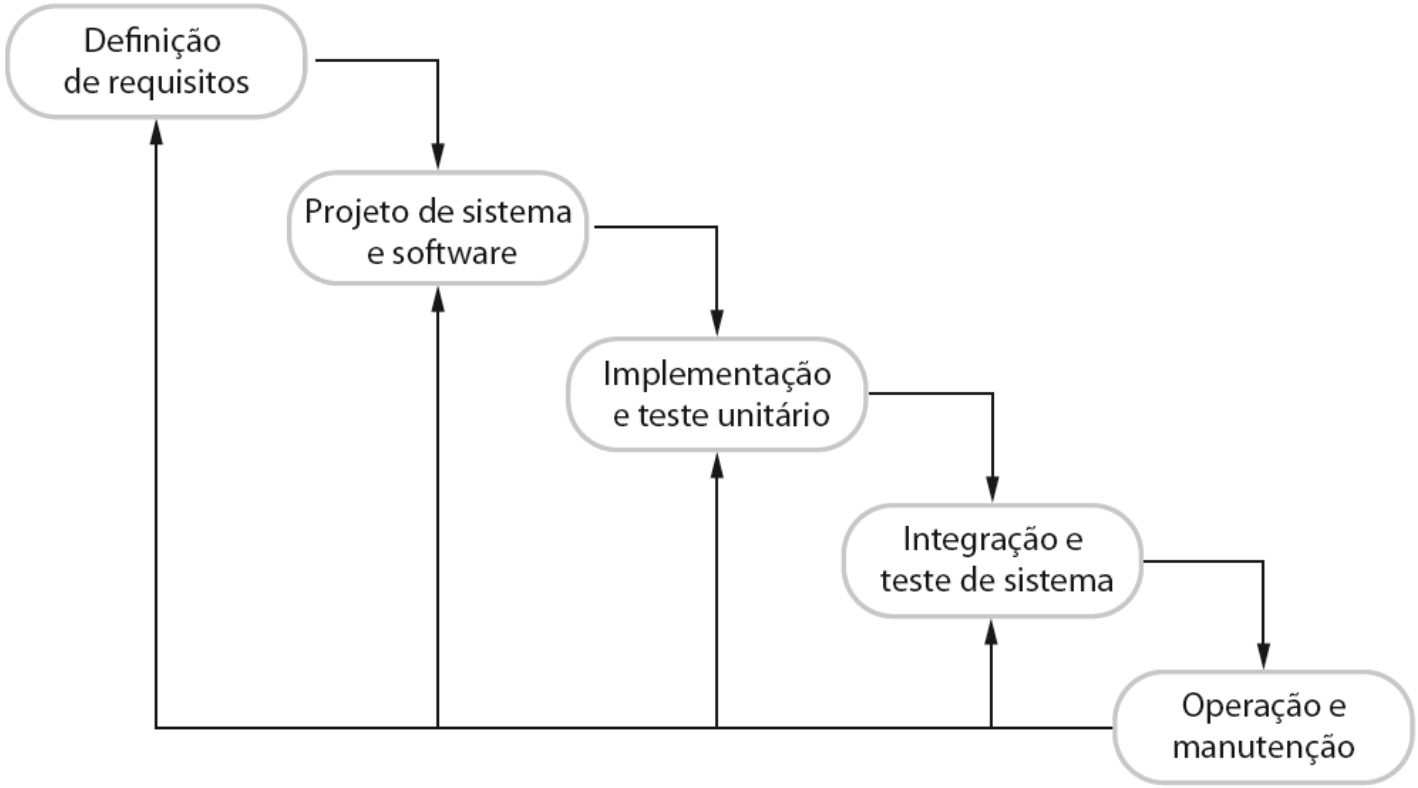
Exemplos de modelos de processo

- **Modelos de processo mais gerais**
  - **Modelo cascata**
  - **Desenvolvimento incremental**
  - **Engenharia de software orientada a reuso**
- Modelos que lidam com mudanças
  - Prototipagem
  - Entrega incremental
  - Modelo espiral

Modelo cascata

- Atividades sequenciais
- Exemplo de um processo dirigido a planos
  - Você deve planejar e programar todas as atividades do processo antes de começar a trabalhar nelas
- Uma fase deve ser terminada para a outra começar
  - Raramente ocorre na prática

Modelo cascata



- Em geral, o resultado de cada estágio é a aprovação de um ou mais documentos (assinados)

Vantagens do modelo cascata

- Documentação rígida (idealmente completa) em cada atividade
  - O estágio seguinte não deve ser iniciado até que a fase anterior seja concluída
- Reflete abordagens adotadas em outras engenharias
  - Gerenciamento mais fácil
- Aderência a outros modelos de processo
  - Pode ser combinado a outros modelos

Desvantagens do modelo cascata

- Projetos reais raramente seguem um fluxo sequencial
- Em geral, é difícil para o cliente estabelecer todos os requisitos à priori
- Difícil se adequar a mudanças inevitáveis de requisitos
- Uma versão executável somente ficará pronta na fase final do projeto

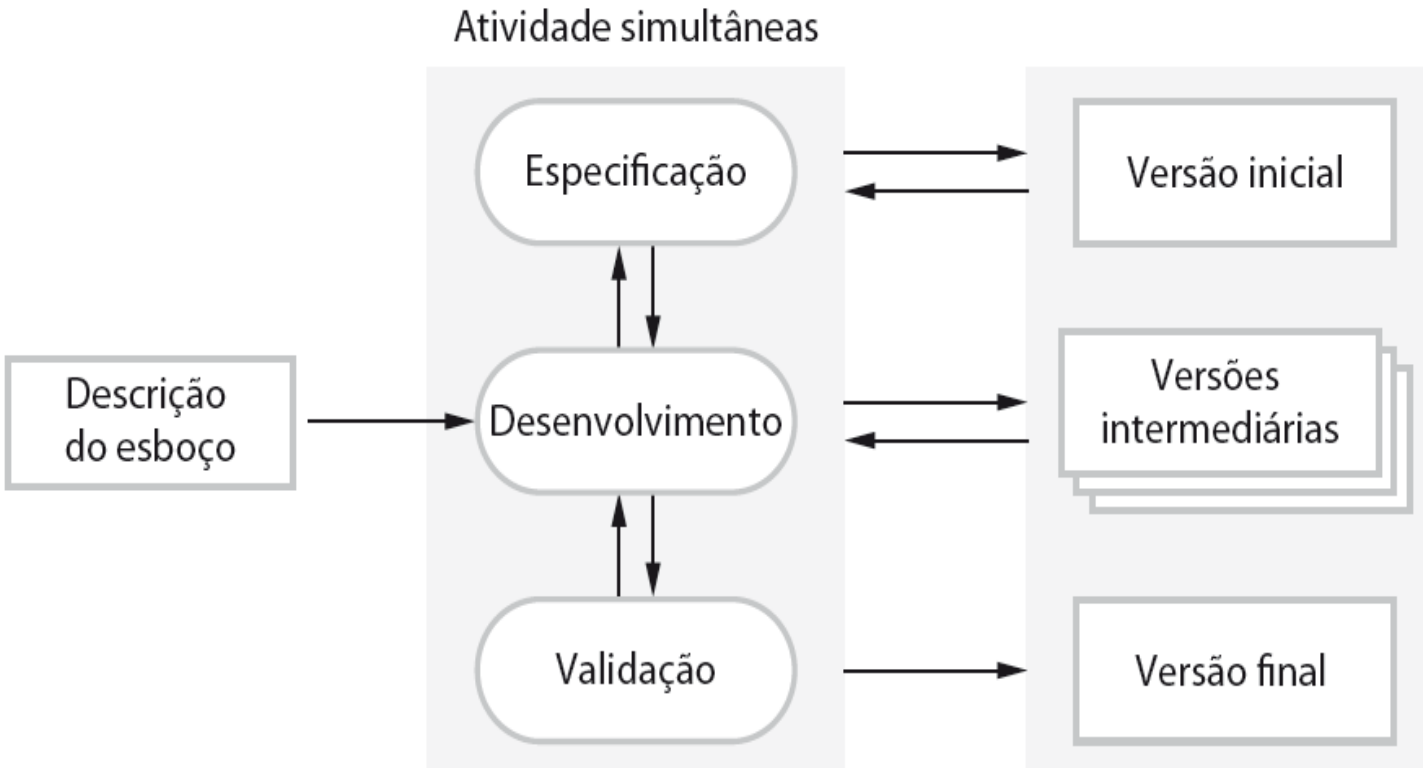
Quando aplicar o modelo cascata?

- Sistemas críticos
- Quando os requisitos são bem compreendidos
- Quando há pouca probabilidade dos requisitos mudarem

Desenvolvimento incremental

- Atividades são intercaladas
  - Desenvolver uma implementação inicial, expô-la aos comentários dos usuários e continuar por meio da criação de várias versões até que um sistema adequado seja desenvolvido
- Objetivo: dar *feedback* rápido ao cliente
- É a parte fundamental das abordagens ágeis
  - É melhor que a abordagem em cascata para a maioria dos sistemas de negócios, *e-commerce* e sistemas pessoais

Desenvolvimento incremental



Desenvolvimento incremental

- Pode ser dirigido a planos, ágil ou uma combinação dessas abordagens
- Dirigido a planos:
  - Os incrementos do sistema são identificados previamente
- Processo ágil:
  - Os incrementos iniciais são identificados, mas o desenvolvimento de incrementos posteriores depende do progresso e das prioridades dos clientes

Vantagens do desenvolvimento incremental

- Custo de acomodar mudanças nos requisitos é reduzido
- Mais fácil obter *feedback* do cliente
- Permite trabalhar com o cliente o entendimento dos requisitos
- Pode-se começar o sistema pelas partes melhor entendidas

Desvantagens do desenvolvimento incremental

- O processo pode não ser muito claro
- A gerência do software é complicada
  - O sistema não é completamente especificado à priori
- A estrutura do produto tende a se corromper com a adição de incrementos
  - O produto final pode se tornar mal estruturado

Desvantagens do desenvolvimento incremental

- Os problemas do desenvolvimento incremental se tornam mais graves em sistemas críticos
- Normalmente não é adequado para sistemas de vida-longa, grandes e complexos, nos quais várias equipes desenvolvem diferentes partes do sistema
  - Esses sistemas necessitam de uma arquitetura estável que deve ser planejada com antecedência e não desenvolvida de forma incremental

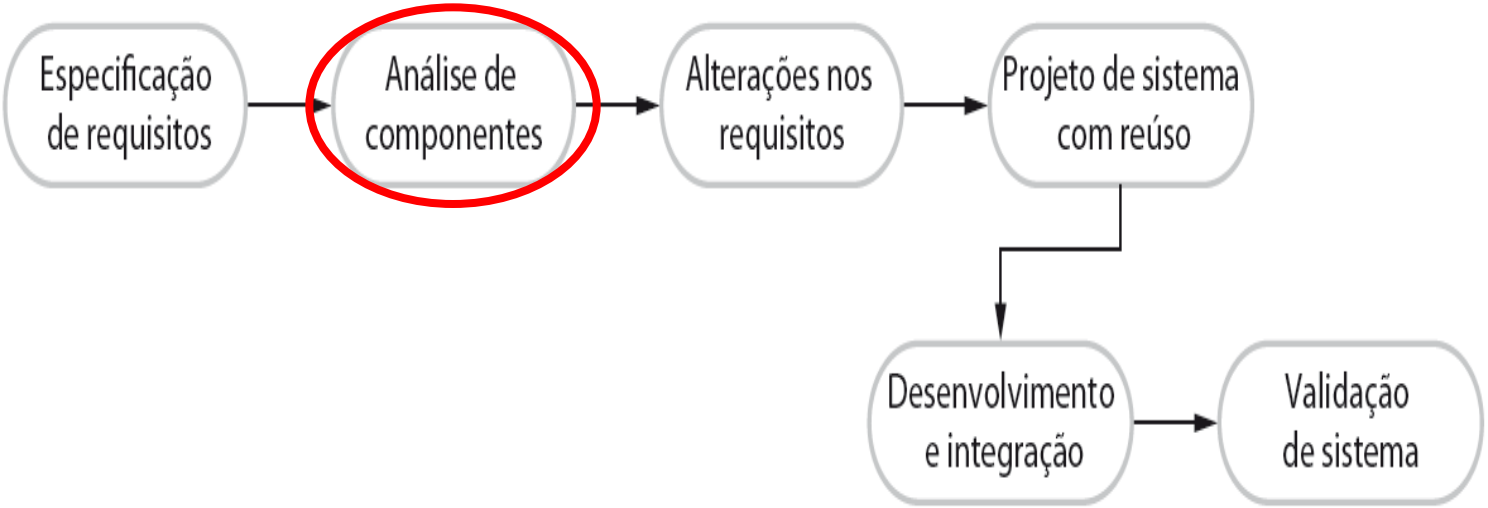
Engenharia de software orientada a reuso

- Na maioria dos projetos de software, há algum reuso de software (mesmo que informalmente)
- Recentemente, processos de desenvolvimento de software com foco no reuso de software existente tornaram-se amplamente usados
- Inspirado na analogia com componentes de hardware
  - Exemplo: componentes elétricos / eletrônicos

Engenharia de software orientada a reuso

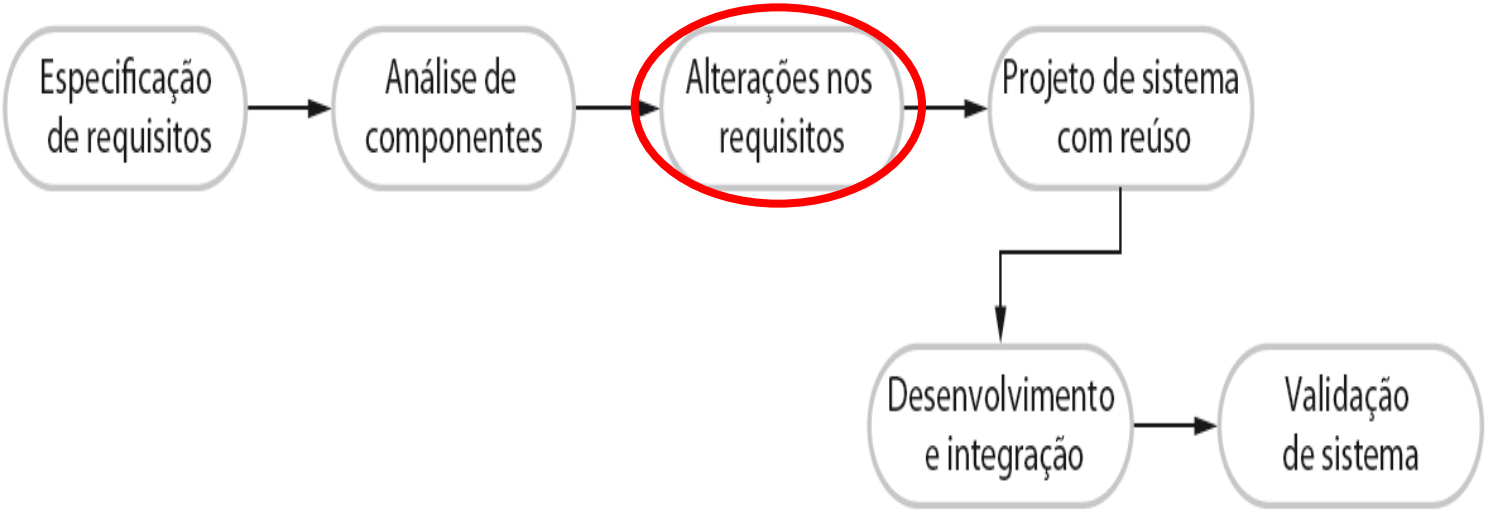
- Baseia-se na existência de um número significativo de componentes reusáveis
- O processo se concentra na integração dos componentes

Engenharia de software orientada a reuso



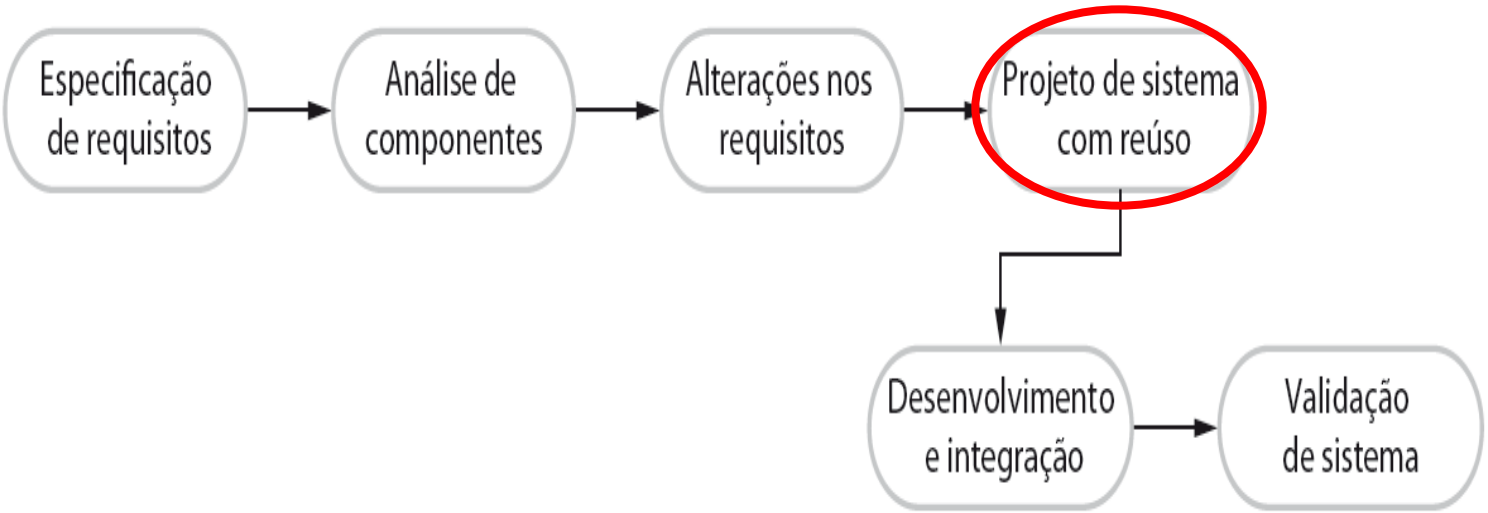
- Análise de componentes
  - Dada uma especificação, encontrar componentes que a atendam

Engenharia de software orientada a reuso



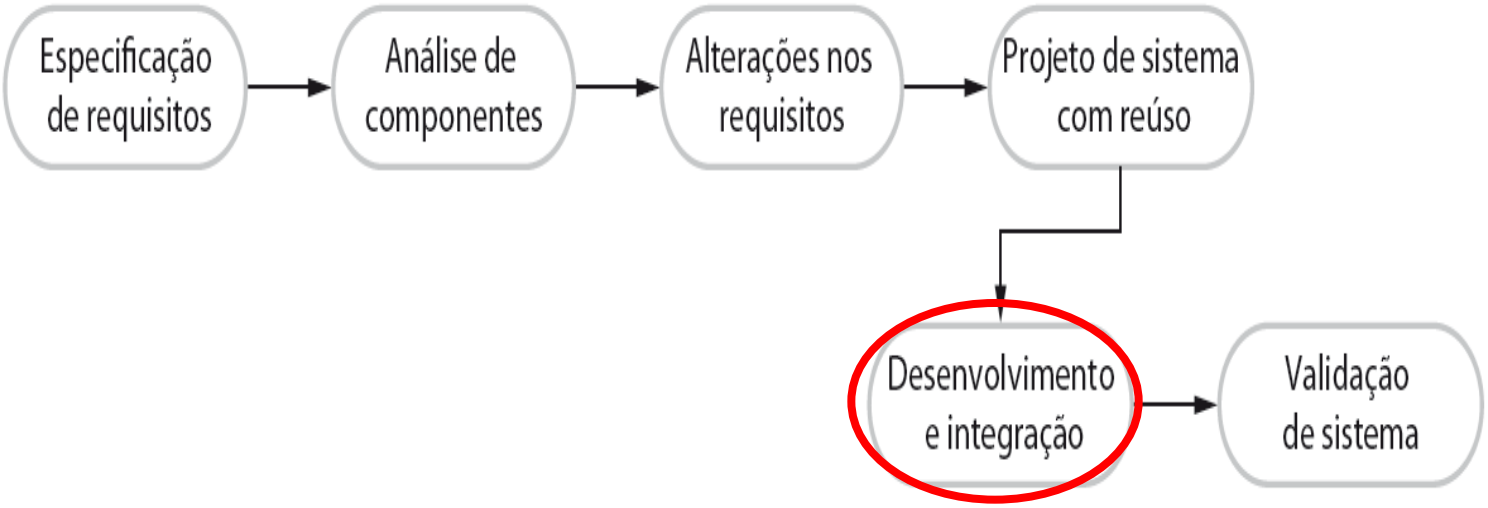
- Alteração nos requisitos
  - Se possível, os requisitos são adaptados aos componentes existentes

Engenharia de software orientada a reuso



- Projeto do sistema com reuso
  - Se necessário, projeta-se novos componentes reusáveis

Engenharia de software orientada a reuso



- Desenvolvimento e integração
  - Desenvolvimento de novos componentes
  - Integração de todos os componentes

Vantagens da engenharia de software orientada a reuso

- Reduz a quantidade de software a ser desenvolvido
- Espera-se reduzir os custos e os riscos
- Espera-se uma entrega do produto mais rápida ao cliente

Desvantagens da engenharia de software orientada a reuso

- Pode-se desenvolver um produto que não atenda aos requisitos do cliente
- Pode ser mais difícil evoluir os sistemas
  - Componentes de terceiros
- A gerência de versões dos componentes pode ser complexa

Qual modelo de processo usar?

- Sistemas críticos
  - Sugerido um modelo de processo baseado em planos mais estruturado e rigoroso, como o modelo cascata

- Sistemas de negócios (requisitos mudam com frequência)
  - Sugerido um modelo de processo ágil e flexível como o desenvolvimento incremental ou o baseado em reuso

Exercícios

1. Justificando sua resposta com base no tipo de sistema a ser desenvolvido, sugira o modelo genérico de processo de software mais adequado para ser usado como base para a gerência do desenvolvimento dos sistemas a seguir:
   a) Um sistema para controlar o antibloqueio de frenagem de um carro.
   b) Um sistema de realidade virtual para dar apoio à manutenção de software.
   c) Um sistema de contabilidade para uma universidade que substitua um sistema já existente.
   d) Um sistema interativo de planejamento de viagens que ajude os usuários a planejar viagens com menor impacto ambiental.

Exercícios

2. Explique por que no modelo de processo baseado em reuso é essencial ter duas atividades distintas de engenharia de requisitos.

Exemplos de modelos de processo

- Modelos de processo mais gerais
  - Modelo cascata
  - Desenvolvimento incremental
  - Engenharia de software orientada a reuso
- **Modelos que lidam com mudanças**
  - **Prototipagem**
  - **Entrega incremental**
  - **Modelo espiral**

Prototipação

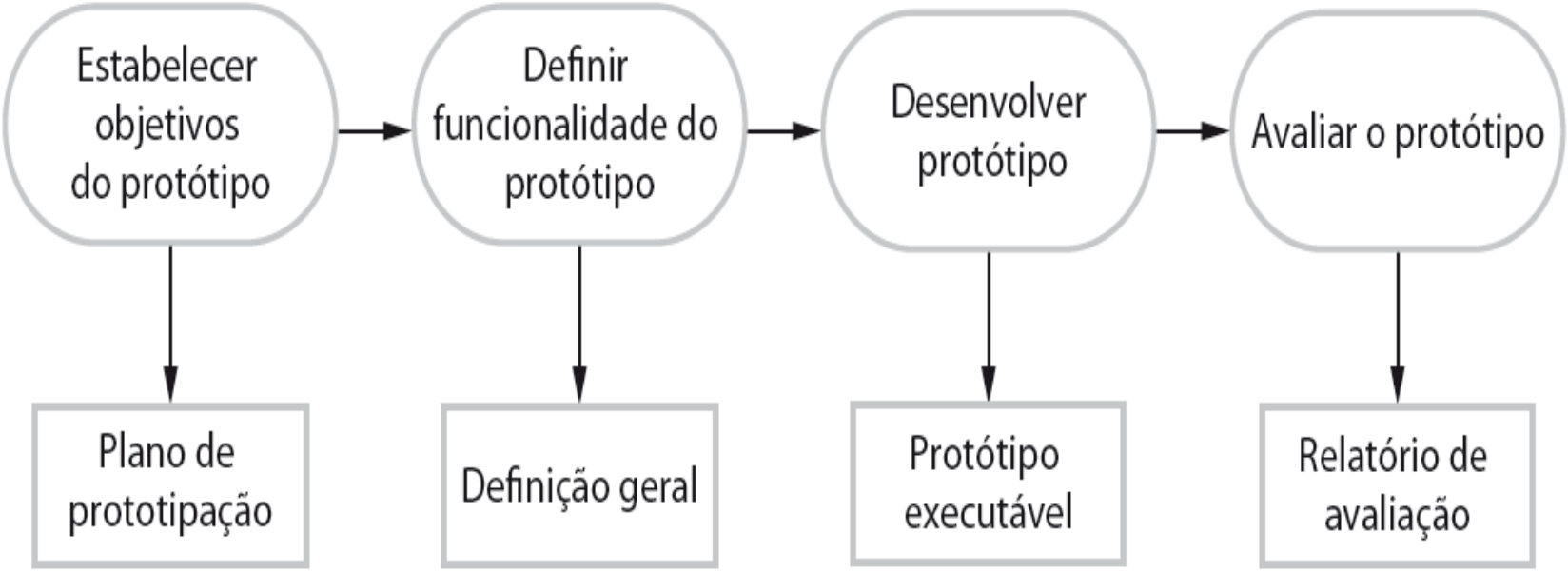
- Protótipo é uma versão inicial de um sistema de software usado para:
  - Demonstrar conceitos
  - Experimentar opções de projeto
  - Descobrir mais sobre o problema e suas possíveis soluções
- Planeja e modela rapidamente um protótipo
  - Mais comum na definição de interfaces com os usuários (telas)
- Começa com os requisitos menos compreendidos
  - Objetivo: entender os requisitos

Prototipação

- O protótipo deveria ser descartado
  - E o sistema re-implementado usando outro processo (exemplo, cascata)
- Principal vantagem
  - Auxilia o engenheiro de software e o cliente a entenderem melhor o que deve ser construído
- É geralmente usada junto com outro modelo de processo

Prototipação

- Modelo de processo para desenvolvimento de protótipos



Prototipação

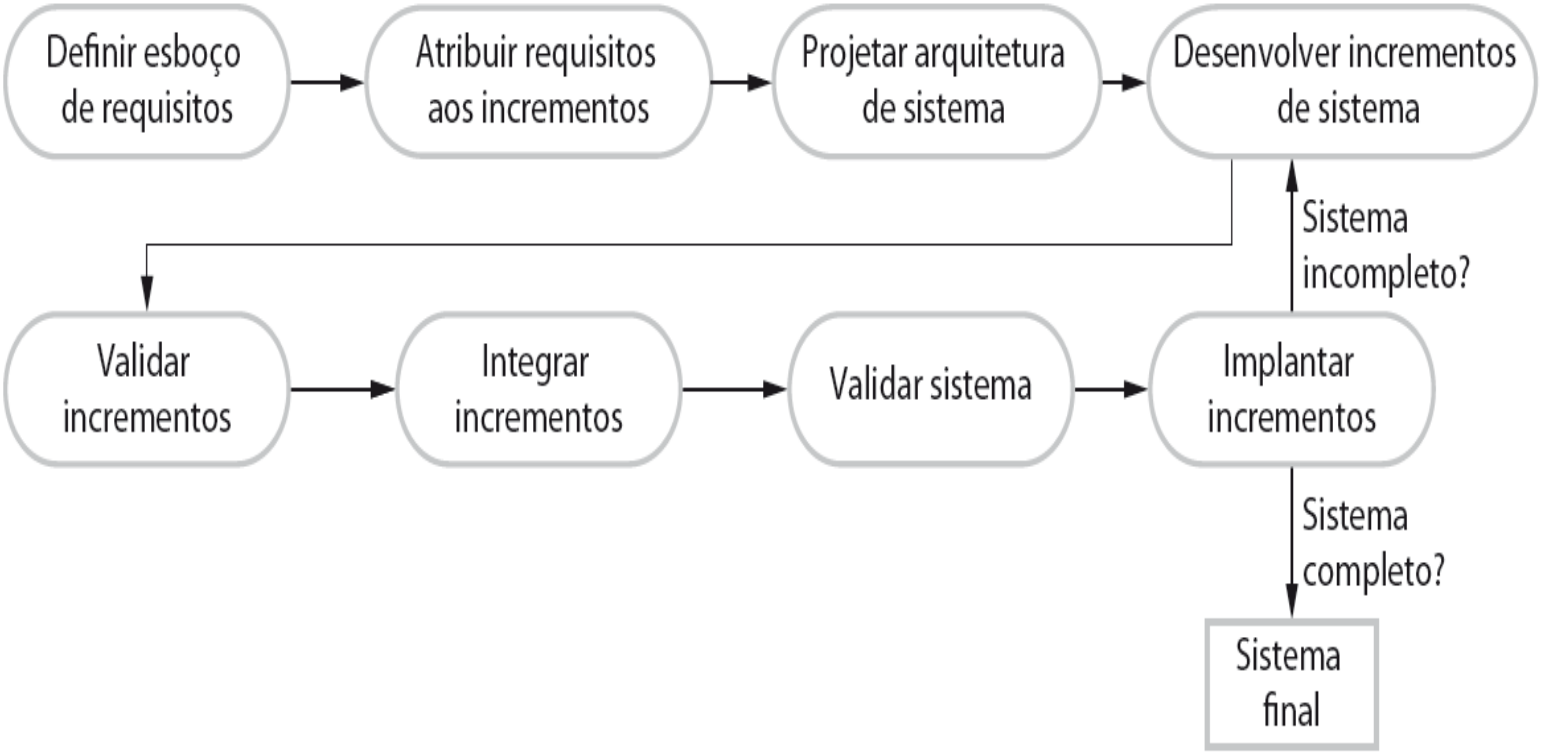
- Ao final do processo de prototipação deve-se:
  - Fazer um planejamento para o treinamento do usuário
  - Comparação dos resultados obtidos com os objetivos do protótipo

Entrega incremental

- Alguns dos incrementos desenvolvidos são entregues ao cliente e implantados para uso em um ambiente operacional
- Os clientes identificam, em linhas gerais, os serviços a serem fornecidos pelo sistema
- Uma série de incrementos de entrega são definidos, com cada incremento proporcionando um subconjunto da funcionalidade do sistema
  - Serviços de mais alta prioridade são implementados e entregues em primeiro lugar

Entrega incremental

- Combina elementos do modelo cascata aplicados de maneira iterativa



Vantagens da entrega incremental

- Os clientes não precisam esperar a entrega final do sistema
  - Ao contrário de protótipos, trata-se de partes do sistema real
  - Eles podem usar o sistema parcial
- Serviços de mais alta prioridade podem ser entregues primeiro
- O risco de falha global do projeto é menor que o modelo cascata

Desvantagens da entrega incremental

- Pode ser difícil definir os recursos comuns e propriedades globais do sistema
- Difícil adoção pelos usuários quando um novo sistema (parcial) irá substituir um sistema antigo (completo)
- Pode causar dificuldades no fechamento do contrato
  - Não há especificação completa a priori

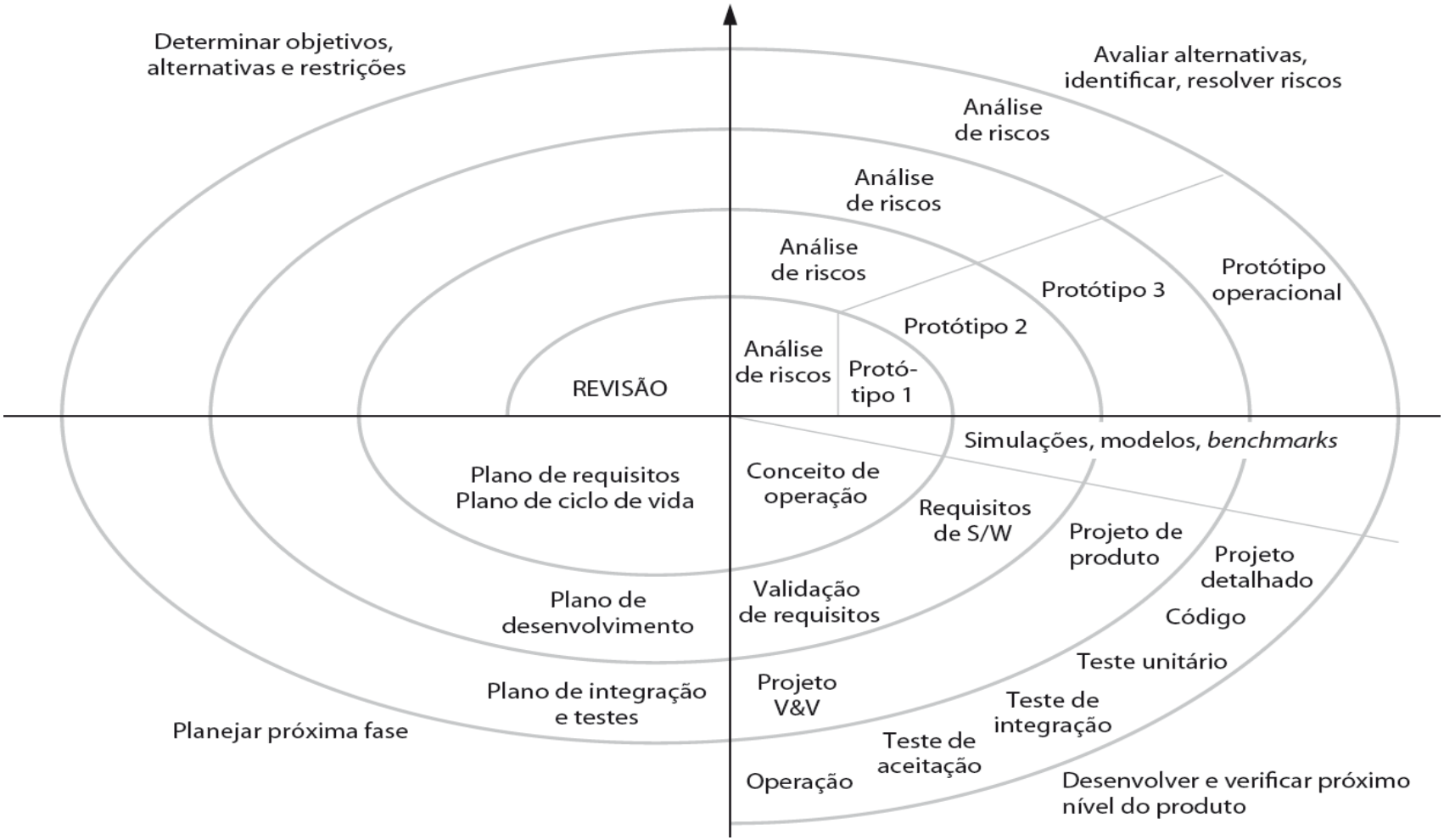
Modelo espiral

- O processo de software é representado como uma espiral, cada volta na espiral representa uma fase do processo de software
  - A volta mais interna pode preocupar-se com a viabilidade do sistema, a volta seguinte com a definição de requisitos, a próxima com o projeto do sistema e assim por diante

Modelo espiral

- Cada volta da espiral é dividida em quatro setores:
  - Definição de objetivos
  - Avaliação e redução de riscos
  - Desenvolvimento e validação
  - Planejamento

Modelo espiral



Modelo espiral

- A principal diferença entre o modelo espiral e outros modelos de processo de software é seu reconhecimento explícito do risco
  - Em cada ciclo as fontes de risco do projeto são identificadas
  - Os riscos devem ser resolvidos por meio de atividades de coleta de informações, como análise mais detalhada, prototipação e simulação

Métodos Ágeis de Desenvolvimento de Software

Métodos ágeis

- Baseiam-se em uma abordagem incremental para a especificação, o desenvolvimento e a entrega do software

Por que processos ágeis?

- As regras de negócios mudam rapidamente
  - O software tem que ser adaptado para as novas regras
- Desenvolvimento e entrega rápida são importantes em mercados competitivos
  - A entrega rápida pode ser tão (ou mais) desejável que a qualidade

Desenvolvimento baseado em planos X ágil

- Baseado em planos:
  - Especificam completamente os requisitos antes de projetar, construir e testar o sistema

- Ágil:
  - Têm por objetivo criar software útil rapidamente
  - Não se preocupam com a documentação completa em todas as fases

De onde vem os métodos ágeis?

- Década de 80
  - A visão é de processos rigorosos para desenvolvimento de software
  - Objetivo: produzir sistemas grandes, complexos e de vida longa
- Década de 90
  - Métodos ágeis ganham força
  - Objetivo: se concentrar no software e não no projeto e documentação

Características gerais

1) Processos ágeis são geralmente iterativos
   - Seguem uma série de incrementos
   - Cada incremento inclui uma nova funcionalidade ao sistema

Características gerais

2) Atividades de especificação, projeto e implementação são feitas em paralelo
   - Especificação não é detalhada
   - Documentação de projeto é mínima ou gerada automaticamente

3) A interface do sistema é criada rapidamente
   - Antes das funcionalidades serem implementadas

Incrementos

- O sistema é desenvolvido por uma série de incrementos
- Usuários finais (ou representante do cliente) participam da especificação e validação de cada incremento

Vantagens

- Entrega acelerada de partes dos serviços ao cliente
  - Partes mais importantes podem ser entregues primeiro
- Engajamento dos usuários finais
  - Maior chance dos usuários ficarem satisfeitos com o produto

Problemas

- Gerenciamento
  - Falta documentação para o gerente
- Fechar o contrato com o cliente
  - Não há especificação completa do sistema
- Validar o sistema
  - A equipe de verificação e validação não tem a especificação
- Modificações contínuas podem corromper a estrutura do sistema

Quando evitar métodos ágeis

- Sistemas grandes e complexos
  - O sistema deve ser gerenciável
- Quando as equipes trabalham em locais distribuídos
  - A comunicação é mais difícil
- Em sistemas críticos
  - Não pode haver falhas
  - Os requisitos devem ser completamente especificados

Manifesto Ágil

- É uma declaração de princípios que fundamentam o desenvolvimento ágil de software
- Passamos a valorizar:
  - **Indivíduos e interações** mais que processos e ferramentas
  - **Software em funcionamento** mais que documentação abrangente
  - **Colaboração com o cliente** mais que negociação de contratos
  - **Responder a mudanças** mais que seguir um plano

http://agilemanifesto.org/

Exemplos e princípios

- Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado
- Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto
- O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face

Resumo dos princípios

Princípios	Descrição
Envolvimento do cliente	Os clientes devem estar intimamente envolvidos no processo de desenvolvimento. Seu papel é fornecer e priorizar novos requisitos do sistema e avaliar suas iterações.
Entrega incremental	O software é desenvolvido em incrementos com o cliente, especificando os requisitos para serem incluídos em cada um.
Pessoas, não processos	As habilidades da equipe de desenvolvimento devem ser reconhecidas e exploradas. Membros da equipe devem desenvolver suas próprias maneiras de trabalhar, sem processos prescritivos.
Aceitar as mudanças	Deve-se ter em mente que os requisitos do sistema vão mudar. Por isso, projete o sistema de maneira a acomodar essas mudanças.
Manter a simplicidade	Focalize a simplicidade, tanto do software a ser desenvolvido quanto do processo de desenvolvimento. Sempre que possível, trabalhe ativamente para eliminar a complexidade do sistema.

Cliente está engajado?

- Envolvimento do cliente
  - É atrativo

- Mas,
  - Depende de um cliente disponível
  - O cliente deve ser capaz de transmitir seu conhecimento a equipe de desenvolvimento

Problemas de comunicação

- Foco na pessoa, não no processo
  - Valorização da equipe

- Mas,
  - Os membros da equipe devem ter perfil adequado para trabalho intenso e colaborativo
  - Exige muita comunicação

Mudanças X simplicidade

- Priorizar mudanças é complicado
  - O que implementar primeiro?
  - Quando tem vários *stakeholders*, quem deve ser priorizado?

- Manter a simplicidade requer trabalho (intelectual) extra
  - Como lidar com a pressão do cronograma de entregas

Desenvolvimento baseado em planos X ágil

- A maioria dos projetos incluem elementos de processos dirigidos a planos e ágeis. Decidir no equilíbrio depende de:

1. **É importante ter uma especificação e projeto bem detalhados antes de passar para a implementação?** Caso seja, provavelmente você precisa usar uma abordagem dirigida a planos.

2. **Uma estratégia de entrega incremental onde você entrega o software para os clientes e recebe *feedback* rápido deles é possível?** Caso seja, considere usar métodos ágeis.

Desenvolvimento baseado em planos X ágil

3. **Qual o tamanho do sistema a ser desenvolvido?** Os métodos ágeis são mais efetivos quando o sistema pode ser desenvolvido com uma equipe pequena que pode se comunicar informalmente. O que pode não ser possível para sistemas grandes que requerem grandes equipes de desenvolvimento, nesses casos, deve ser usada uma abordagem dirigida a planos.

4. **Que tipo de sistema está sendo desenvolvido?** Abordagens dirigidas a planos podem ser necessárias para sistemas que requerem muita análise antes da implementação (ex. sistema que opere em tempo real com requisitos de temporização complexos).

Desenvolvimento baseado em planos X ágil

5. **Qual é o tempo de vida esperado para o sistema?** Sistemas com longo tempo de vida podem precisar de mais documentação de projeto para comunicar as intenções originais dos desenvolvedores do sistema para a equipe de suporte.

6. **Quais tecnologias estão disponíveis para manter o desenvolvimento do sistema?** Métodos ágeis dependem de boas ferramentas para acompanhar um sistema em evolução.

Desenvolvimento baseado em planos X ágil

7. **Como está organizada a equipe de desenvolvimento?** Se a equipe de desenvolvimento está distribuída ou se parte do desenvolvimento está sendo terceirizado você pode precisar desenvolver documentos de projeto para que haja comunicação entre as equipes de desenvolvimento.

8. **Existem questões culturais ou organizacionais que podem afetar o desenvolvimento do sistema?** As organizações tradicionais de engenharia têm uma cultura de desenvolvimento dirigido a planos, o que é padrão em engenharia.

Desenvolvimento baseado em planos X ágil

9. **O quão bons são os projetistas e os programadores da equipe de desenvolvimento?** É dito que os métodos ágeis requerem um nível de habilidade mais alto do que as abordagens dirigidas a planos, nas quais os programadores simplesmente traduzem um projeto detalhado em código.

10. **O sistema está sujeito a regulamentação externa?** Se o sistema precisa ser aprovado por um regulador externo (ex. O FAA aprova softwares críticos para a operação de um avião) então provavelmente requisitaram a você a produção de documentação detalhada como parte da documentação de segurança do sistema.

Exercícios

3. Explique por que, em sistemas complexos, as mudanças são inevitáveis. Exemplifique as atividades de processo de software que ajudam a prever as mudanças e fazer com que o software seja desenvolvido mais tolerante a mudanças.
4. Explique como os princípios básicos do manifesto ágil levam ao desenvolvimento e implantação de software acelerados.
5. Quando você não recomendaria o uso de método ágil para o desenvolvimento de um sistema de software?

Programação Extrema (XP)

- Proposta a partir de boas práticas de desenvolvimento iterativo
- Propõe o envolvimento do cliente ao extremo
  - O cliente (ou seu representante) deve estar disponível durante todo o desenvolvimento
- Programadores trabalham em pares

Dos requisitos aos testes

- Os requisitos são escritos como cenários (estórias do usuário)
  - Estórias determinam algo que o sistema precisa fazer e são descritas pelo cliente
  - Estas estórias são implementadas diretamente por um conjunto de tarefas

- Para cada tarefa, é desenvolvido um conjunto de testes
  - Testes são feitos antes da implementação

Exemplo de estória

Prescrição de medicamentos

Kate é uma médica que deseja prescrever medicamentos para um paciente de uma clínica. O prontuário do paciente já está sendo exibido em seu computador, assim, ela clica o campo 'medicação' e pode selecionar 'medicação atual', 'nova medicação', ou 'formulário'.

Se ela selecionar 'medicação atual', o sistema pede que ela verifique a dose. Se ela quiser mudar a dose, ela altera esta e em seguida, confirma a prescrição.

Se ela escolher 'nova medicação', o sistema assume que ela sabe qual medicação receitar.
Ela digita as primeiras letras do nome do medicamento. O sistema exibe uma lista de possíveis fármacos que começam com essas letras. Ela escolhe a medicação requerida e o sistema responde, pedindo-lhe para verificar se o medicamento selecionado está correto.
Ela insere a dose e, em seguida, confirma a prescrição.

Se ela escolhe 'formulário', o sistema exibe uma caixa de busca para o formulário aprovado.
Ela pode, então, procurar pelo medicamento requerido. Ela seleciona um medicamento e é solicitado que verifique se a medicação está correta. Ela insere a dose e, em seguida, confirma a prescrição.

O sistema sempre verifica se a dose está dentro da faixa permitida. Caso não esteja, Kate é convidada a alterar a dose.

Após Kate confirmar a prescrição, esta será exibida para verificação. Ela pode escolher 'OK' ou 'Alterar'. Se clicar em 'OK', a prescrição fica gravada nos bancos de dados da auditoria.
Se ela clicar em 'Alterar', reinicia o processo de 'Prescrição de Medicamentos'.

Exemplos de cartões de tarefa

Tarefa 1: Alterar dose de medicamentos prescritos

Tarefa 2: Seleção de formulário

Tarefa 3: Verificação de dose

A verificação da dose é uma precaução de segurança para verificar se o médico não receitou uma dose perigosamente pequena ou grande.

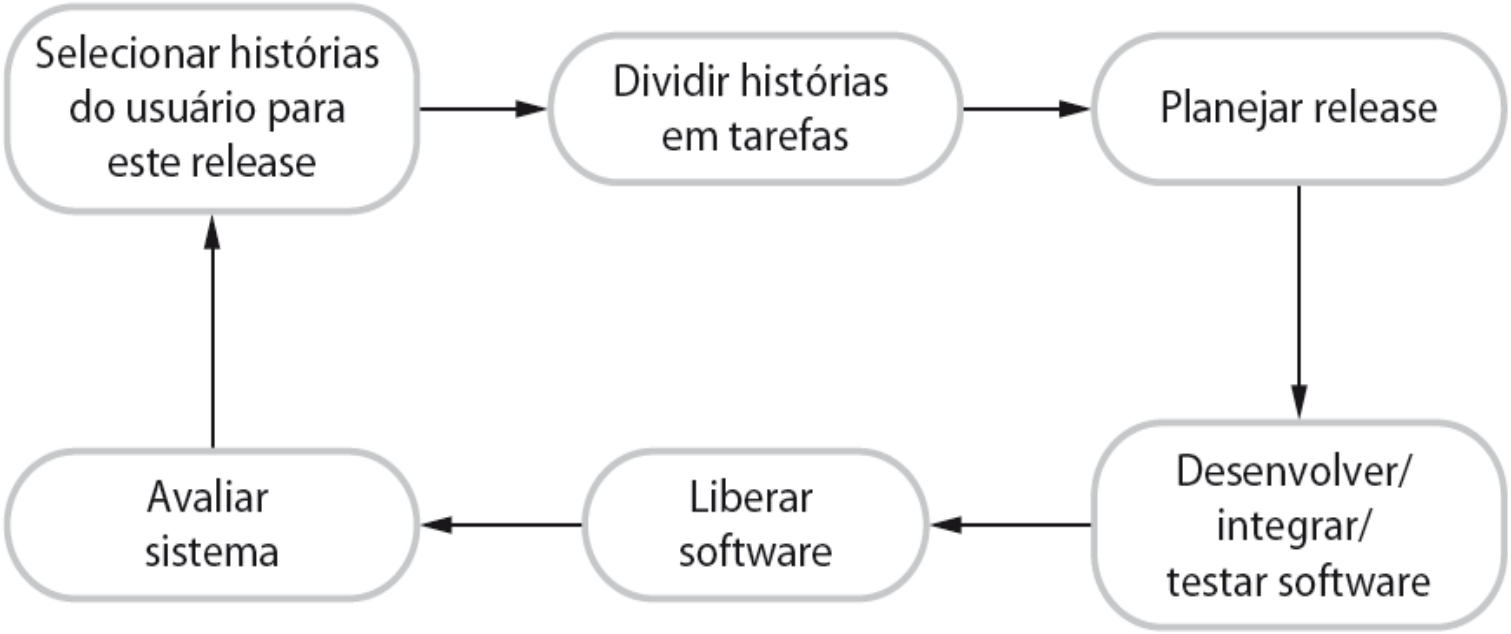
Usando o ID do formulário para o nome do medicamento genérico, procure o formulário e obtenha a dose mínima e máxima recomendada.

Verifique a dose mínima e máxima prescrita. Caso esteja fora da faixa, emita uma mensagem de erro dizendo que a dose está muito alta ou muito baixa.

Caso esteja dentro da faixa, habilite o botão 'Confirmar'.

Modelo de processo XP para produção de um incremento do sistema

- O espaço de tempo entre *releases* é curto



Práticas da programação extrema

Princípio ou prática	Descrição
Planejamento incremental	Os requisitos são gravados em cartões de história e as histórias que serão incluídas em um release são determinadas pelo tempo disponível e sua relativa prioridade. Os desenvolvedores dividem essas histórias em 'Tarefas'. Veja os quadros 3.1 e 3.2.
Pequenos *releases*	Em primeiro lugar, desenvolve-se um conjunto mínimo de funcionalidades útil, que fornece o valor do negócio. *Releases* do sistema são frequentes e gradualmente adicionam funcionalidade ao primeiro *release*.
Projeto simples	Cada projeto é realizado para atender às necessidades atuais, e nada mais.
Desenvolvimento *test-first*	Um *framework* de testes iniciais automatizados é usado para escrever os testes para uma nova funcionalidade antes que a funcionalidade em si seja implementada.
Refatoração	Todos os desenvolvedores devem refatorar o código continuamente assim que encontrarem melhorias de código. Isso mantém o código simples e manutenível.

Práticas da programação extrema

Princípio ou prática	Descrição
Programação em pares	Os desenvolvedores trabalham em pares, verificando o trabalho dos outros e prestando apoio para um bom trabalho sempre.
Propriedade coletiva	Os pares de desenvolvedores trabalham em todas as áreas do sistema, de modo que não se desenvolvam ilhas de *expertise*. Todos os conhecimentos e todos os desenvolvedores assumem responsabilidade por todo o código. Qualquer um pode mudar qualquer coisa.
Integração contínua	Assim que o trabalho em uma tarefa é concluído, ele é integrado ao sistema como um todo. Após essa integração, todos os testes de unidade do sistema devem passar.
Ritmo sustentável	Grandes quantidades de horas-extra não são consideradas aceitáveis, pois o resultado final, muitas vezes, é a redução da qualidade do código e da produtividade a médio prazo.
Cliente no local	Um representante do usuário final do sistema (o cliente) deve estar disponível todo o tempo à equipe de XP. Em um processo de Extreme Programming, o cliente é um membro da equipe de desenvolvimento e é responsável por levar a ela os requisitos de sistema para implementação.

Pequenos *releases*

- Novas versões do sistema podem ser compiladas várias vezes por dia
  - Testes unitários automatizados devem ser executados após cada compilação
- Incrementos são entregues ao cliente a cada duas semanas

Refatorações

- XP prega que modelar o sistema para mudanças futuras é um esforço inútil
- Refatorações são constantemente aplicadas para permitir adaptações

Testes em XP

- XP enfatiza mais as atividades de teste que outros métodos ágeis

- Práticas
  - Desenvolver primeiro os testes
  - Codificação incremental a partir dos testes
  - Envolvimento do usuário na escrita e validação dos testes
  - Uso de ferramentas para testes automatizados

- Testes de aceitação também são incrementais

Descrição de caso de teste

Teste 4: Verificação de dose

Entrada:
1. Um número em mg representando uma única dose da medicação.
2. Um número que representa o número de doses únicas por dia.

Testes:
1. Teste para entradas em que a dose única é correta, mas a frequência é muito alta.
2. Teste para entradas em que a única dose é muito alta e muito baixa.
3. Teste para entradas em que a dose única x frequência é muito alta e muito baixa.
4. Teste para entradas em que a dose única x frequência é permitida.

Saída:
Mensagem de OK ou erro indicando que a dose está fora da faixa de segurança.

Desvantagens de testar primeiro

- Programadores preferem programar do que testar
  - Teste podem ser mal feitos ou incompletos
- Alguns testes são difíceis de escrever
  - Pode ser tão difícil quando implementar
- É difícil avaliar a abrangência dos testes

Programação por pares

- Dois programadores sentam juntos na frente de um mesmo computador

- Os pares nem sempre são os mesmos
  - A alocação dinâmica dos pares é sugerido, pois favorece a propriedade coletiva do código

Vantagens da programação por pares

- Responsabilidade comum
  - As acertos e falhas são de responsabilidade de toda a equipe
- Processo informal de revisão
  - Enquanto um programa, o parceiro revisa informalmente o código
- Favorece a melhoria da qualidade
  - Os parceiros discutem oportunidades para refatorações

Scrum

- Método ágil geral, mas seu foco está no gerenciamento do desenvolvimento iterativo
  - Scrum não é sigla, mas algumas empresas usam letras maiúsculas (SCRUM)

- Objetivo
  - Oferecer uma forma de gerenciar métodos ágeis

Adaptação constante

- Scrum propõe uma forma flexível de trabalho
  - Trocas de equipes ou membros da equipe
  - Adaptações de cronograma e orçamento
  - Uso de variadas ferramentas de desenvolvimento ou linguagens de programação

- Adequado para requisitos que sofrem mudanças constantes

Semelhanças com o XP

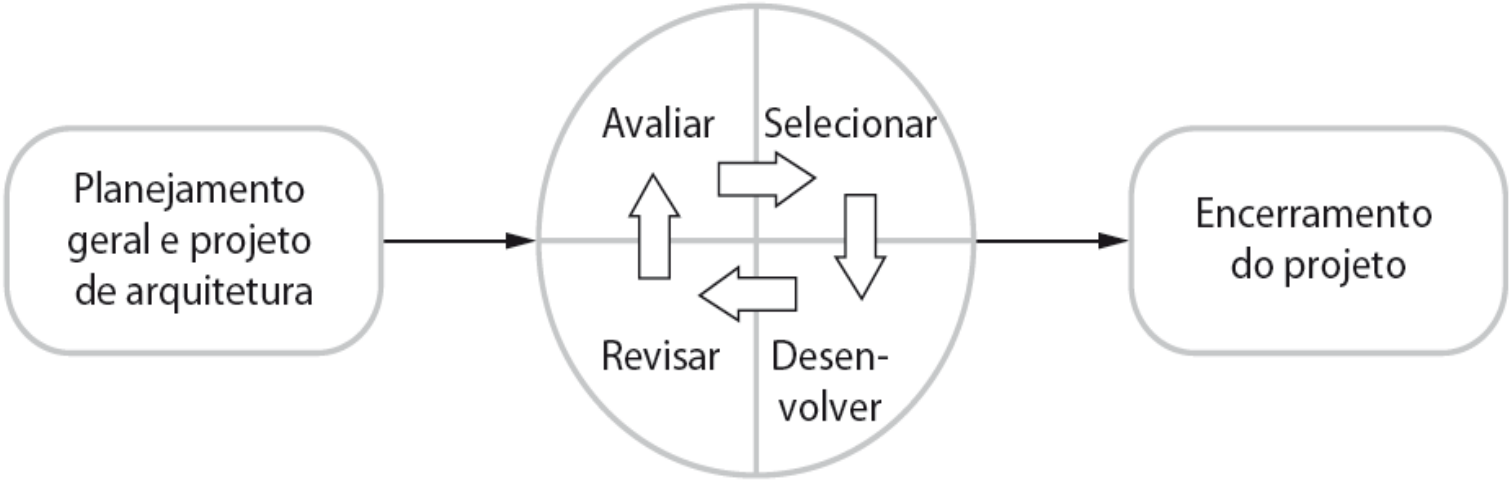
- Equipes pequenas
- Trabalho com requisitos instáveis ou desconhecidos
- Iterações curtas
- Envolvimento do cliente

Scrum X XP

- Scrum divide o desenvolvimento em ciclos de até 30 dias (*sprints*)
- Define papéis para os membros da equipe
  - *Scrum master*, *product owner*, gerentes, etc.
- Reuniões diárias para acompanhamento

Ciclo de processo

- Planejamento geral
- Ciclos de *sprint*
- Encerramento do projeto



Ciclo *Sprint*

Planejamento

- Os requisitos iniciais são descritos e armazenados no *product backlog*
- Requisitos são ordenados e agrupados em *sprint backlog*
- Uma estimativa inicial de esforço é feita
- Uma arquitetura inicial para o sistema é proposta

Ciclos de *sprint*

- O software é desenvolvido em ciclos, chamados *sprints*
  - Um *sprint* varia entre duas semanas a um mês
  - Reuniões diárias são feitas para acompanhar os problemas e o andamento das tarefas

- Cada *sprint* segue atividades semelhantes a de um processo tradicional
  - Análise de requisitos, projeto, implementação e testes

Entrega final

- É feita a integração e testes finais
- Preparada a documentação dos usuários
- A equipe se reúne para
  - Analisar os resultados do projeto
  - Identificar problemas que podem ser corrigidos em projetos futuros
- Demonstrar e entregar o produto final ao cliente

Definição de papéis

- O papéis dos *stakeholders* caem em duas categorias **porcos** e **galinhas**

Papéis de porcos

- Os porcos são os que estão comprometidos com os objetivos do projeto todo
  - *Scrum master*
  - Equipe
  - *Product owner*

- Estes definem o sucesso da implantação e continuidade do Scrum em uma empresa

Papéis de porcos

- *Scrum Master* (facilitador)
  - Líder da equipe
  - Tem a função de remover impedimentos para que a equipe atinja os objetivos
  - Garante que o processo está sendo usado e impões a aplicação de regras

- *Team*
  - Responsável por entregar o produto
  - Formado tipicamente por 5 a 9 pessoas

Papéis de porcos

- *Product Owner*
  - Representa a voz do cliente
  - Pode ser o próprio cliente ou alguém que tem a visão dele e que ele confia para administrar seu projeto
  - Nos projetos Scrum, ele tem uma importância tão grande quanto a própria equipe ou o *Scrum Master*

Papéis de galinhas

- Representantes do cliente
  - Pessoas que criam o ambiente para implantação do produto na organização
- Outros *Stakeholders*
  - Representam as várias pessoas envolvidas com o projeto
  - Podem ser clientes ou fornecedores

As reuniões diárias

- No início de cada dia de um *sprint*, é feita uma reunião (*daily scrum*)
  - Sempre começa na hora
  - Duração de precisamente 15 minutos
  - Devem ocorrer no mesmo local e no mesmo horário

- Todos são bem vindos, mas somente os porcos falam

Perguntas das reuniões diárias

- Cada membro da equipe deve responder às seguintes perguntas
  - O que eu fiz desde ontem?
  - O que planejo fazer hoje?
  - Algo me impediu de atingir meu objetivo?

- Caso tenha tido algum impedimento, o *scrum master* é responsável por solucioná-lo

Reuniões de planejamento do *sprint*

- Ocorre no início de cada *sprint* (entre 14 a 30 dias)
  - Selecionar o trabalho a ser feito no *sprint* dentre os definidos no *product backlog*
  - Preparar o *sprint backlog* que detalha o cronograma e responsabilidades
- É limitada a um período de 8 horas
  - 4 horas para priorização (*product owner*)
  - 4 horas para planejamento (*team*)

Reuniões de revisão do *sprint*

- Revisar o trabalho que foi completado (ou não) no *sprint*
- Apresentar o resultado do *sprint* aos *stakeholders* (demo)
- É limitado a um período de 4 horas

Exercícios

6. Programação extrema expressa os requisitos dos usuários como estória, com cada estória escrita em um cartão. Discuta as vantagens e desvantagens dessa abordagem para a descrição de requisitos.

7. Sugira razões pelas quais a taxa de produtividade de programadores que trabalham em pares pode ser mais que a metade da taxa de produtividade de dois programadores que trabalham individualmente.

Exercícios

8. Tem-se sugerido que um dos problemas de se ter um usuário participando de uma equipe de desenvolvimento de software é que eles “se tornam nativos”, ou seja, adotam a perspectiva da equipe de desenvolvimento e perdem de vista as necessidades de seus colegas usuários. Sugira maneiras de evitar esse problema e discuta as vantagens e desvantagens de cada abordagem.