Engenharia de Software I

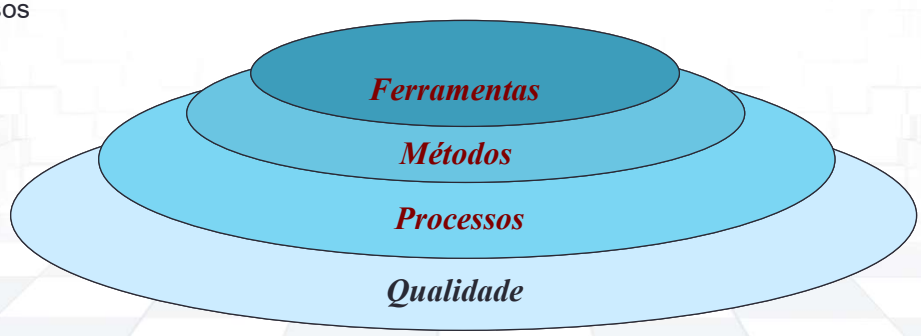
Modelos de Processo de Software

Marcelo Buscioli Tenorio

1

Engenharia de Software
Considerações

- Abrange um conjunto de três elementos fundamentais, tendo como base a qualidade do software:
  - Ferramentas
  - Métodos
  - Processos



2

Engenharia de Software
Ferramentas

- Dão suporte automatizado aos métodos
- Existem atualmente ferramentas para sustentar cada um dos métodos
- Quando as ferramentas são integradas é estabelecido um sistema de suporte ao desenvolvimento de software chamado *CASE – Computer Aided Software Engineering*

3

Engenharia de Software
Métodos

- Proporcionam os detalhes de como fazer para construir o software:
  - Planejamento e estimativa de projeto
  - Análise de requisitos de software e de sistemas
  - Projeto da estrutura de dados
  - Codificação
  - Teste
  - Manutenção

4

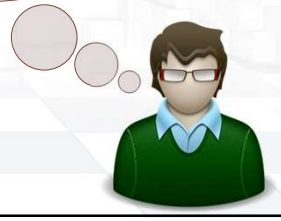
Engenharia de Software
Processos

- *Constituem o elo de ligação entre os métodos e ferramentas:*
  - *Sequência em que os métodos serão aplicados*
  - *Produtos que se exige que sejam entregues*
  - *Controles que ajudam assegurar a qualidade e coordenar as alterações*
  - *Marcos de referência que possibilitam administrar o progresso do software*

5

Engenharia de Software

Não posso programar um software de
qualquer maneira...
Preciso seguir algumas etapas
em seu ciclo de vida…



6

Engenharia de Software
Ciclo de Vida

- Conjunto de etapas que envolve métodos, ferramentas e processos
- Essas etapas são conhecidas como componentes do **Ciclo de Vida do Software**

7

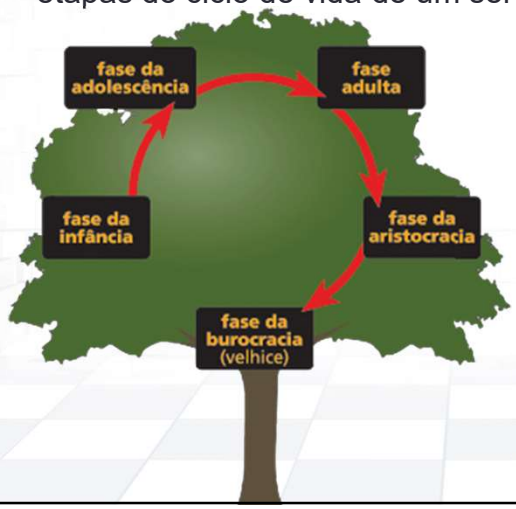
Ciclo de Vida do Software [1/2]

- O ciclo de vida de um software (*software lifecycle*) designa todas as etapas da concepção, do desenvolvimento, à extinção
- Essa segmentação tem por objetivo definir pontos intermediários que permitam checar a conformidade do sistema com as necessidades expressas no escopo do projeto e verificar o processo de desenvolvimento

8

Ciclo de Vida do Software [2/2]

- O ciclo de vida, em sistemas informatizados, tem as mesmas etapas do ciclo de vida de um ser humano [Adizes, 1999]



Podemos fazer uma analogia do ciclo de vida de um software com o ciclo de vida de uma pessoa, como veremos a seguir...

9

Analogia: Software/Pessoa

Fase de Namoro [Adizes, 1999]

- O namoro é considerado o primeiro estágio do desenvolvimento, quando o sistema ainda nem nasceu e existe apenas como uma ideia
  - Ainda não existe fisicamente, é apenas uma possibilidade
- Trata-se, portanto, de um período em que se fala muito e se age pouco
  - O analista de sistemas tem, nessa etapa, uma função muito importante: entender o que o cliente necessita (levantamento de requisitos)
    - e a partir daí, criar um compromisso com ele

10

Analogia: Software/Pessoa

Fase da Infância [Adizes, 1999]

- Também chamada de sistema-criança, conta com poucos controles formalizados e muitas falhas a serem transformadas em virtudes
  - Normalmente, o sistema é precário, faltam registros e informações, há resistência das pessoas em fazer reuniões de aprimoramento
    - Alguns chegam a acreditar que o sistema não vai funcionar

11

Analogia: Software/Pessoa

Fase da Adolescência [Adizes, 1999]

- É o estágio do renascimento
  - É eliminada grande parte dos erros encontrados na fase anterior
- Essa transição é caracterizada por conflitos e inconsistências, muitas vezes causados pelos próprios usuários, os quais ainda não se comprometem a realizar as interações pertinentes
- Mesmo percebendo a necessidade de delegar autoridade, mudar metas e liderança, os responsáveis enfrentam dificuldades, pois muitos usuários ainda acreditam que o antigo sistema era melhor

12

Analogia: Software/Pessoa

Fase Adulta [Adizes, 1999]

- A estabilidade, ou fase adulta, é o início do estágio de envelhecimento do sistema
  - Quando começa a se tornar obsoleto e surgem outros melhores
- Um sintoma visível é a perda de flexibilidade
  - Todos começam a achar que ele não funciona tão bem, atribuindo-lhe falhas
- É um estágio marcado pelo fim do crescimento e início do declínio

13

Analogia: Software/Pessoa

Fase da Aristocracia [Adizes, 1999]

- A aristocracia (meia-idade ou melhor idade) é a fase da vaidade, do vestir-se bem e de falar bem
  - A ênfase está em como as coisas são feitas e não no porquê [MALUCHE, 2000]
- É iniciada a etapa de declínio total do sistema, na qual o nível de inovação é baixo e tudo deixa a desejar

14

Analogia: Software/Pessoa

Fase da Burocracia [Adizes, 1999]

- Na fase da burocracia (velhice) o sistema perde a funcionalidade e a elasticidade
  - Com isso, ninguém mais tem confiança nele
- Muitos percebem a situação, mas ninguém faz nada, culpando o sistema por todos os erros e falhas na organização
  - O declínio se intensifica e, mesmo que permaneça em uso por alguns anos, a decadência prossegue, até a morte do sistema [ADIZES, 1999]

15

Processo de Software

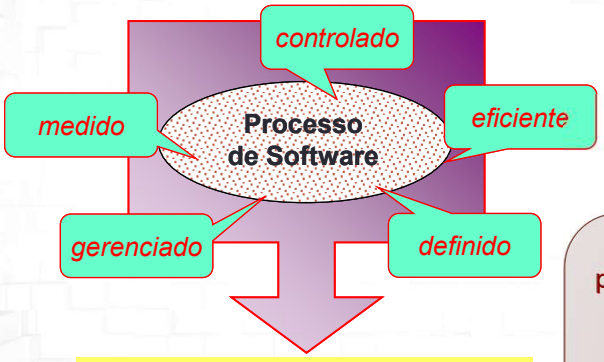
Processo de Software com Qualidade [1/2]

- Para que um processo de software tenha qualidade, ele deve:
  - Ser definido
  - Ser eficiente
  - Ser controlado
  - Ser medido
  - Ser gerenciado

16

Processo de Software

Processo de Software com Qualidade [2/2]



Modelos de Processo de Software

Utilizamos modelos de processo de software para que um processo de software possa ser definido, eficiente, controlado, medido e gerenciado.
Tais modelos representam o ciclo de vida do software.

17

Processo de Software

Modelos de Processo de Software [1/2]

- Procuram descrever formalmente e de maneira organizada todas as etapas (e suas respectivas atividades) que devem ser seguidas para a obtenção segura de um produto de software
  - A definição dessas etapas permite estabelecer pontos de controle para a avaliação da qualidade e da gestão do projeto
    - Auxiliam o controle e a coordenação de um projeto de software

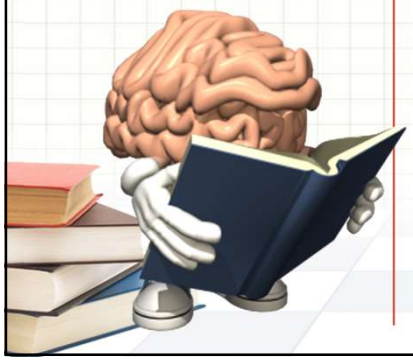
18

Processo de Software
Modelos de Processo de Software [2/2]

- Existem vários modelos de processo de software
  - ou paradigmas de engenharia de software ou modelos de ciclo de vida
  - Mais conhecidos:
    - Cascata, Prototipação e Espiral
- Cada um representa uma tentativa de colocar ordem em uma atividade inerentemente caótica
- São selecionados com base
  - na natureza do projeto e da aplicação
  - nos métodos e ferramentas a serem utilizados
  - nos controles e produtos que precisam ser entregues

19

Tópicos



- Modelos Tradicionais
  - Modelo Cascata
  - Modelo Prototipação
  - Modelo RAD
  - Modelos Evolutivos
    - Modelo Incremental
    - Modelo Espiral
  - Técnicas de 4ª Geração
- Modelos Ágeis (Engenharia de Software II)
  - Scrum
  - XP
  - etc

20

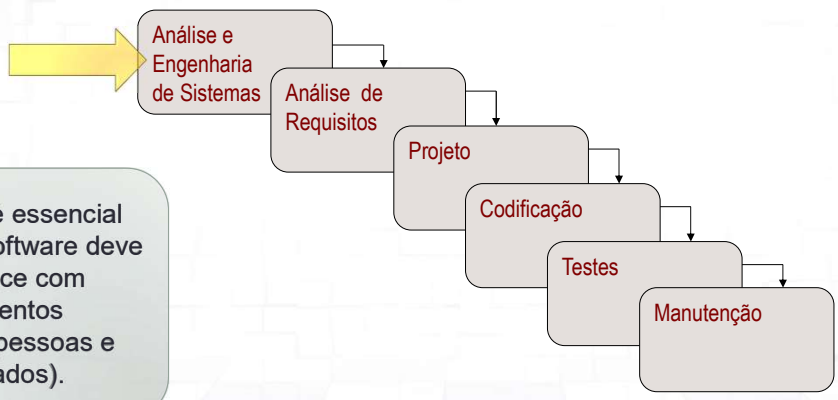
Cascata (Waterfall)

- Modelo mais antigo e o *mais amplamente usado* da engenharia de software
- Também conhecido como Ciclo de Vida Clássico e Modelo Sequencial Linear
- Modelado em função do ciclo da engenharia convencional
- Requer uma abordagem sistemática, sequencial ao desenvolvimento de software

21

Cascata
Fases [1/6]

Envolve a coleta de requisitos em nível do sistema, com uma pequena quantidade de projeto e análise de alto nível.



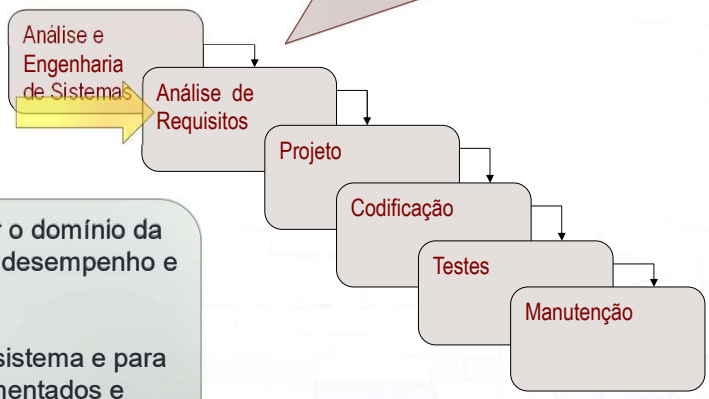
Esta visão é essencial quando o software deve fazer interface com outros elementos (hardware, pessoas e banco de dados).

O resultado de uma fase se constitui na entrada da outra

22

Cascata
Fases [2/6]

O processo de coleta dos requisitos é intensificado e concentrado **especificamente no software**.



Deve-se compreender o domínio da informação, a função, desempenho e interfaces exigidos.

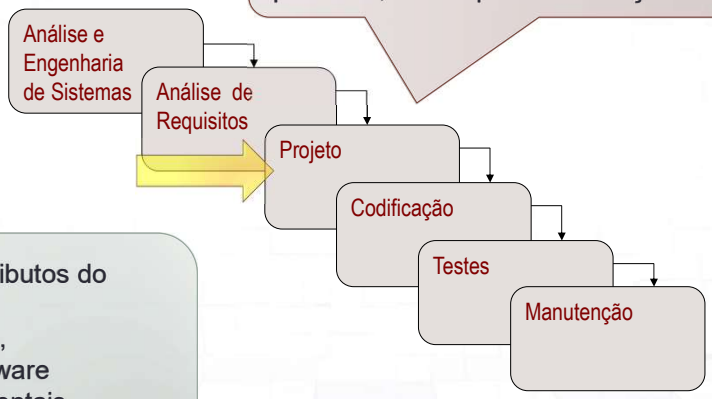
Os requisitos (para o sistema e para o software) são documentados e revistos com o cliente

O resultado de uma fase se constitui na entrada da outra

23

Cascata
Fases [3/6]

Tradução dos requisitos do software para um conjunto de representações que podem ser avaliadas quanto à qualidade, antes que a codificação se inicie.

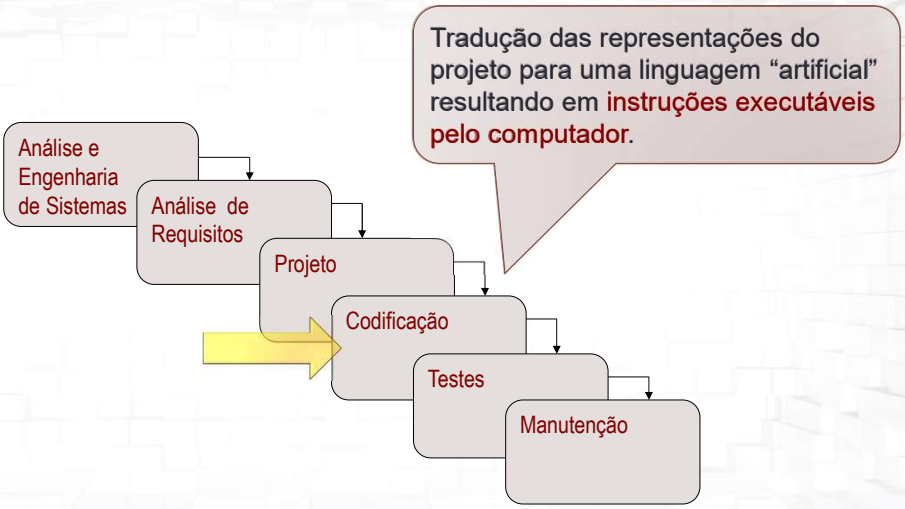


Concentra-se em 4 atributos do programa:
- Estrutura de Dados,
- Arquitetura de Software
- Detalhes Procedimentais
- Caracterização de Interfaces

O resultado de uma fase se constitui na entrada da outra

24

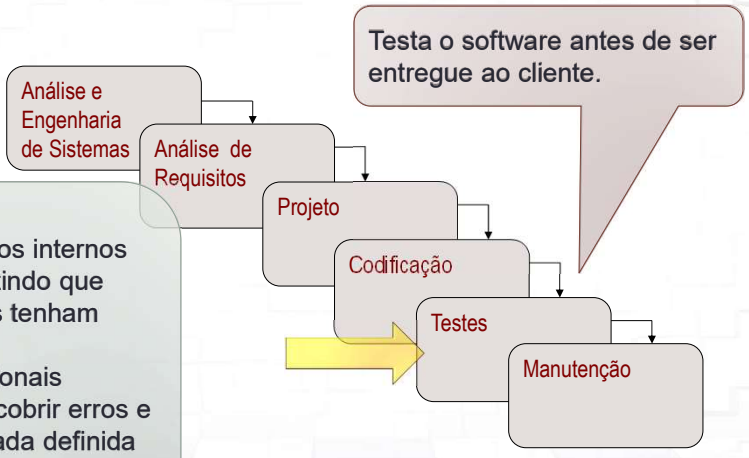
Cascata
Fases [4/6]



O resultado de uma fase se constitui na entrada da outra

25

Cascata
Fases [5/6]

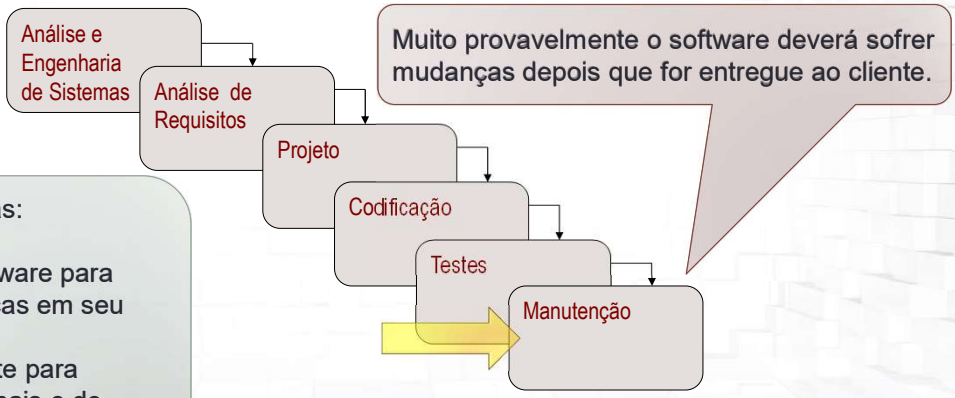


Concentra-se:
- Nos aspectos lógicos internos do software, garantindo que todas as instruções tenham sido testadas.
- Nos aspectos funcionais externos, para descobrir erros e garantir que a entrada definida produza resultados que concordem com os esperados.

O resultado de uma fase se constitui na entrada da outra

26

Cascata
Fases [6/6]



Muito provavelmente o software deverá sofrer mudanças depois que for entregue ao cliente.

Causas das mudanças:
- Erros.
- Adaptação do software para acomodar mudanças em seu ambiente externo.
- Exigência do cliente para acréscimos funcionais e de desempenho.

O resultado de uma fase se constitui na entrada da outra

27

Cascata
Problemas

- Projetos reais raramente seguem o fluxo sequencial que o modelo propõe
- Logo no início é difícil estabelecer explicitamente todos os requisitos. No começo dos projetos sempre existe uma incerteza natural
- O cliente deve ter paciência. Uma versão executável do software só fica disponível no final do ciclo de vida.

28

Cascata
Considerações Finais

- Embora o Ciclo de Vida Clássico tenha fragilidades, ele é significativamente melhor do que uma abordagem casual ao desenvolvimento de software
- O modelo Cascata trouxe contribuições importantes para o processo de desenvolvimento de software:
  - Imposição de disciplina, planejamento e gerenciamento
  - A implementação do produto deve ser postergada até que os objetivos tenham sido completamente entendidos

29

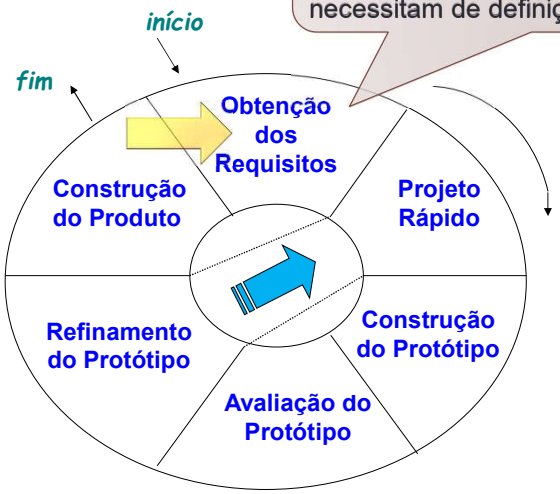
Prototipação

- Objetivo de entender os requisitos do usuário e, assim, obter uma melhor definição dos requisitos do sistema
- Possibilita que o desenvolvedor crie um modelo do software que deve ser construído
- Idealmente, o modelo (protótipo) serve como um mecanismo para identificar os requisitos de software
- Apropriado para quando o cliente não definiu detalhadamente os requisitos
  - Definiu um conjunto de objetivos gerais para o software, mas não identificou requisitos de entrada, processamento e saída com detalhes

30

Prototipação
Fases [1/6]

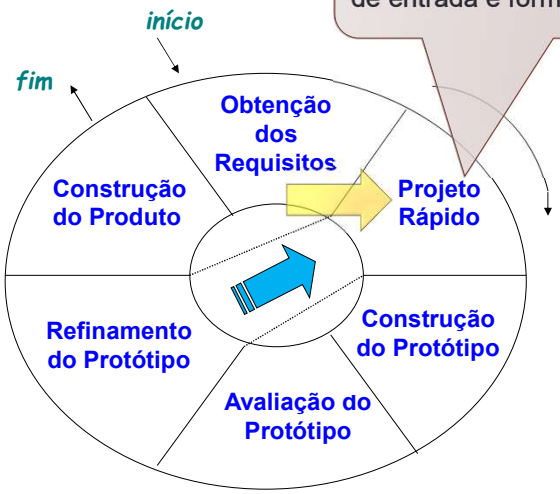
Desenvolvedor e cliente definem os objetivos gerais do software, identificam quais requisitos são conhecidos e as áreas que necessitam de definições adicionais.



31

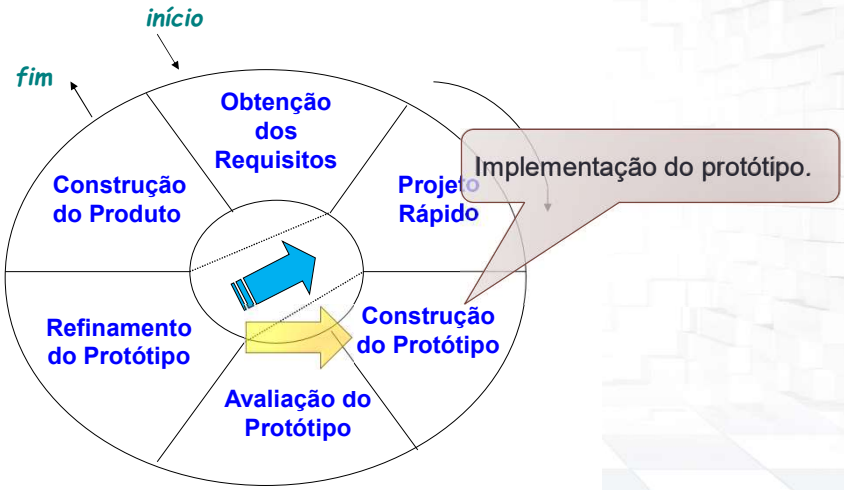
Prototipação
Fases [2/6]

Representação dos aspectos do software que são visíveis ao usuário (abordagens de entrada e formatos de saída).



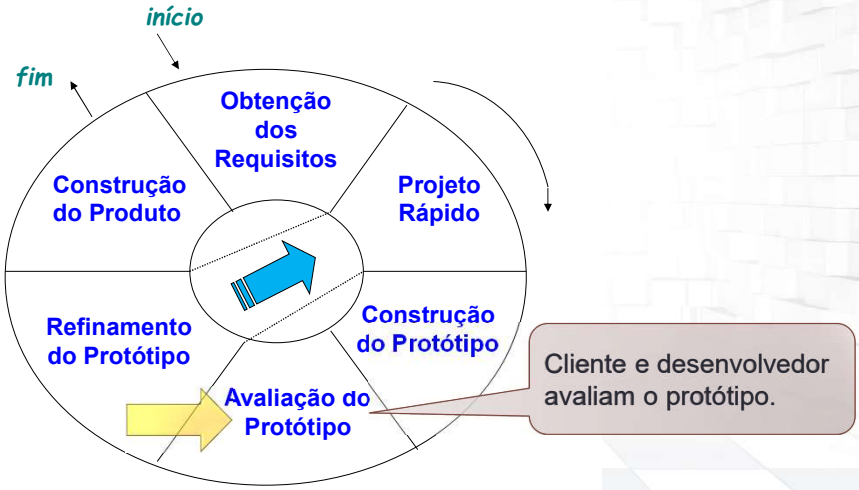
32

Prototipação
Fases [3/6]



33

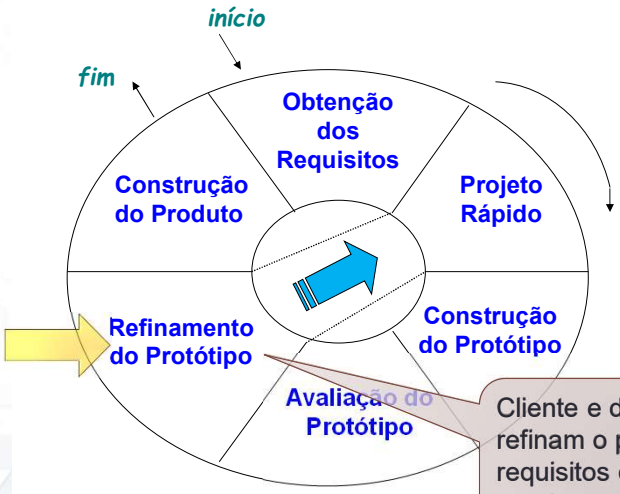
Prototipação
Fases [4/6]



34

Prototipação
Fases [5/6]

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



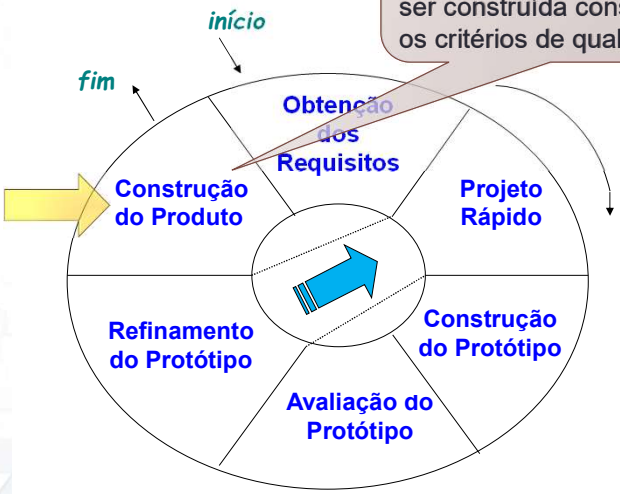
Cliente e desenvolvedor refinam o protótipo e os requisitos do software a ser desenvolvido.

35

Prototipação
Fases [6/6]

O Prototipação não indica como construir o produto.
Podemos mesclar aqui fases de outros modelos, como as de Projeto, Codificação, Testes e Manutenção do Cascata.

Identificados os requisitos, o protótipo deve ser descartado e a versão de produção deve ser construída considerando os critérios de qualidade.



36

Prototipação
Problemas

- Cliente não sabe que, para o software (protótipo) que ele vê não foi considerado, durante o desenvolvimento, a qualidade global e a manutenibilidade a longo prazo
- Cliente não aceita bem a ideia que a versão final do software vai ser construída e "força" a utilização do protótipo como produto final
- Desenvolvedor frequentemente faz uma implementação comprometida (utilizando o que está disponível) com o objetivo de produzir rapidamente um protótipo
  - Com o passar do tempo ele se familiariza com essas escolhas e esquece que elas não são apropriadas para o produto final

37

Prototipação
Considerações Finais

- Ainda que possam ocorrer problemas, a prototipação é um ciclo de vida eficiente
- A chave é a definição das regras de negócio logo no início
- O cliente e o desenvolvedor devem ambos concordar que o protótipo seja construído para servir como um mecanismo a fim de definir os requisitos

38

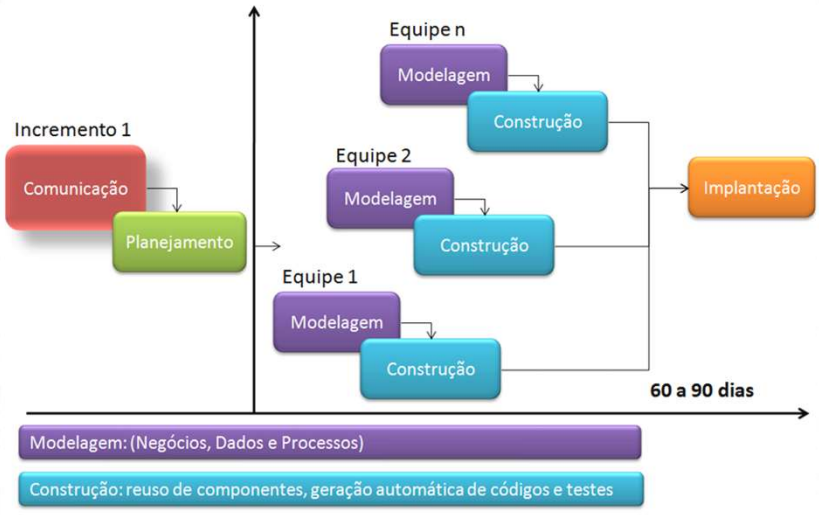
RAD

- *RAD* (*Rapid Application Development*) é um modelo iterativo e incremental que enfatiza um ciclo de desenvolvimento extremamente curto
- O desenvolvimento rápido é obtido usando uma abordagem de construção baseada em componentes

39

RAD

Modelo [Pressman 2010]



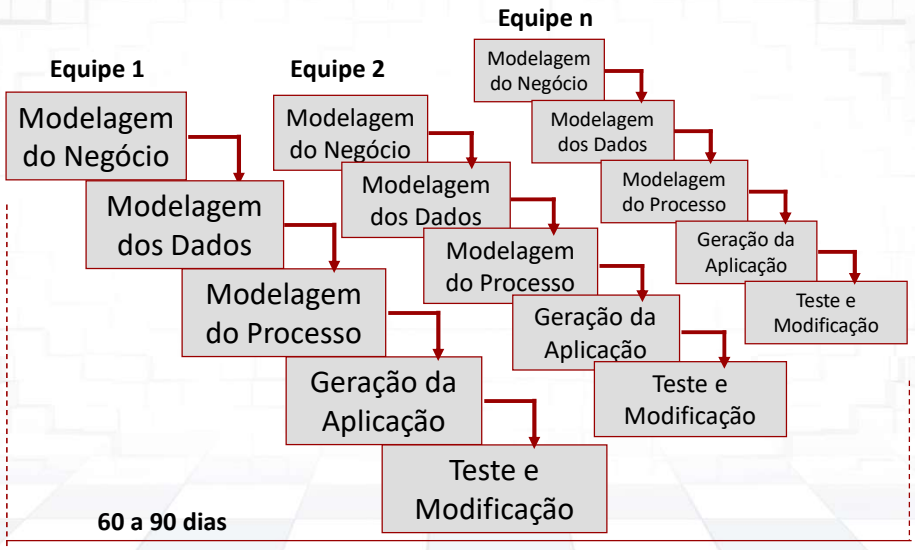
40

RAD
Considerações

- Os requisitos devem ser bem entendidos e o alcance do projeto restrito
- O modelo *RAD* é usado principalmente para aplicações de **sistema de informação**
  - Sistema que manipula dados e gera informação
  - Abrange pessoas, máquinas, e/ou métodos organizados para coletar, processar, transmitir e disseminar dados que representam informação para o usuário e/ou cliente
- Cada função principal pode ser direcionada para uma equipe *RAD* separada e então integrada para formar o todo

41

RAD
Fases



O sistema é dividido em módulos independentes

42

RAD
Modelagem do Negócio

- São levantados os processos de negócios os quais o sistema oferecerá suporte
- O fluxo de informações entre as funções de negócio é modelado de modo a responder às seguintes questões:
  - Que informação direciona o processo de negócio?
  - Que informação é gerada?
  - Quem a gera?
  - Para onde vai a informação?
  - Quem a processa?

43

RAD
Modelagem dos Dados

- Responde a um conjunto de questões específicas que são relevantes a qualquer aplicação
- O fluxo de informação definido na fase de modelagem de negócio é refinado para extrair:
  - Os principais objetos de dados a serem processados pelo sistema
  - Qual a composição de cada um dos objetos de dados
  - Onde costumam ficar
  - Qual a relação entre eles
  - Quais as relações entre os objetos e os processos que os transformam

Entidades que representam um conjunto de atributos. Ex: pessoa, pagamento, carro etc.

Ex: uma pessoa pode possuir um carro, onde o relacionamento *possuir* conota uma conexão específica entre pessoa e carro

44

RAD
Modelagem do Processo

- Os objetos de dados definidos na modelagem de dados são transformados para se obter o fluxo necessário para implementar uma função do negócio
- Descrições do processamento são criadas para adicionar, modificar, descartar ou recuperar um objeto de dados

45

RAD
Geração da Aplicação

- O *RAD* considera o uso de **técnicas de quarta geração**
  - Ferramentas de alto nível
- Trabalha com a reutilização de componentes de programa existentes quando possível
  - ou cria componentes reutilizáveis
- São utilizadas ferramentas automatizadas para facilitar a construção do software
  - Ex: Visual Studio, Eclipse, Android Studio, Xcode, VS Code, etc

46

RAD
Teste e Modificação

- Como o RAD enfatiza o reuso, muitos componentes já estão testados, reduzindo o tempo total de teste
  - Todavia os novos componentes devem ser testados e todas as interfaces devem ser exaustivamente exercitadas

47

RAD
Vantagens

- Permite o desenvolvimento rápido e/ou a prototipagem de aplicações
- Criação e reutilização de componentes
- Desenvolvimento conduzido em um nível mais alto de abstração
- Visibilidade mais cedo (protótipos)
- Maior flexibilidade
  - desenvolvedores podem reprojetar praticamente a vontade
- Grande redução de codificação manual
  - Wizards
- Envolvimento maior do usuário
- Provável custo reduzido
  - tempo é dinheiro e também devido ao reuso
- Aparência padronizada
  - APIs e outros componentes reutilizáveis permitem uma aparência consistente

48

RAD
Desvantagens

- Exige recursos humanos suficientes para todas as equipes
- Exige que desenvolvedores e clientes estejam comprometidos com as atividades de "fogo-rápido" a fim de terminar o projeto num prazo curto
- Mais difícil de acompanhar o projeto
  - Não existe os marcos clássicos
- Pode acidentalmente levar ao retorno das práticas caóticas no desenvolvimento

RAD
Considerações Finais

- Nem todos os tipos de aplicação são apropriadas para o *RAD*:
  - Deve ser possível a modularização efetiva da aplicação
  - Se alto desempenho é uma característica e o desempenho é obtido sintonizando as interfaces dos componentes do sistema, a abordagem *RAD* pode não funcionar

Modelos Evolutivos

- Existem situações em que a engenharia de software necessita de um modelo de processo que possa acomodar um produto que evolui com o tempo:
  - Quando os requisitos de produto e de negócio mudam conforme o desenvolvimento procede
  - Quando de uma data de entrega apertada (mercado)
    - impossível a conclusão de um produto completo
  - Quando um conjunto de requisitos importantes é bem conhecido, porém os detalhes ainda devem ser definidos

51

Modelos Evolutivos
Considerações

- Modelos evolutivos são **iterativos**:
  - Possibilitam o desenvolvimento de versões cada vez mais completas do software
  - Exemplos: Modelo Incremental e Modelo Espiral

52

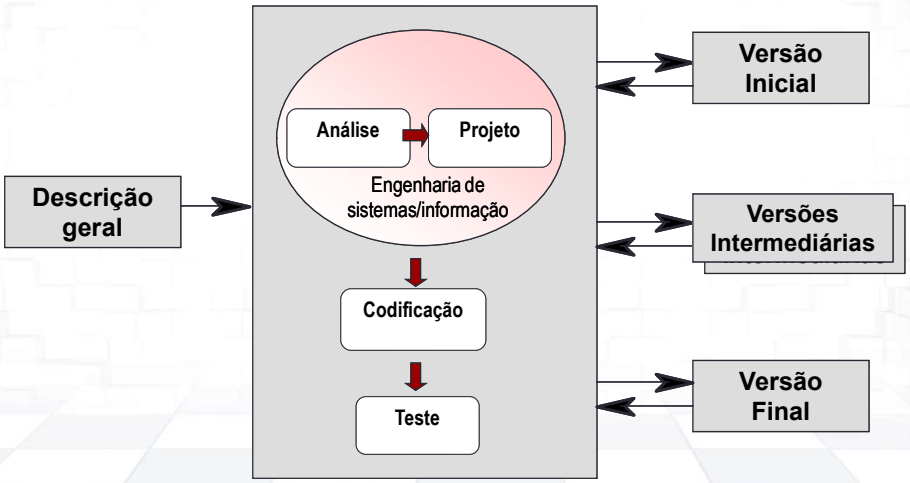
Modelo Incremental

- O modelo incremental combina elementos do modelo cascata (aplicado repetidamente) com a filosofia iterativa da prototipação
- O objetivo é trabalhar junto do usuário para descobrir seus requisitos, de maneira incremental, até que o produto final seja obtido

53

Modelo Incremental

Incrementos [Sommerville]



54

Modelo Incremental
Considerações

- A versão inicial é frequentemente o núcleo do produto
  - A parte mais importante
  - A evolução acontece quando novas características são adicionadas à medida que são sugeridas pelo usuário
- Este modelo é importante quando é difícil estabelecer a *priori* uma especificação detalhada dos requisitos
- É mais apropriado para sistemas pequenos
- As novas versões podem ser planejadas de modo que os riscos técnicos possam ser administrados
  - Ex.: disponibilidade de determinado hardware

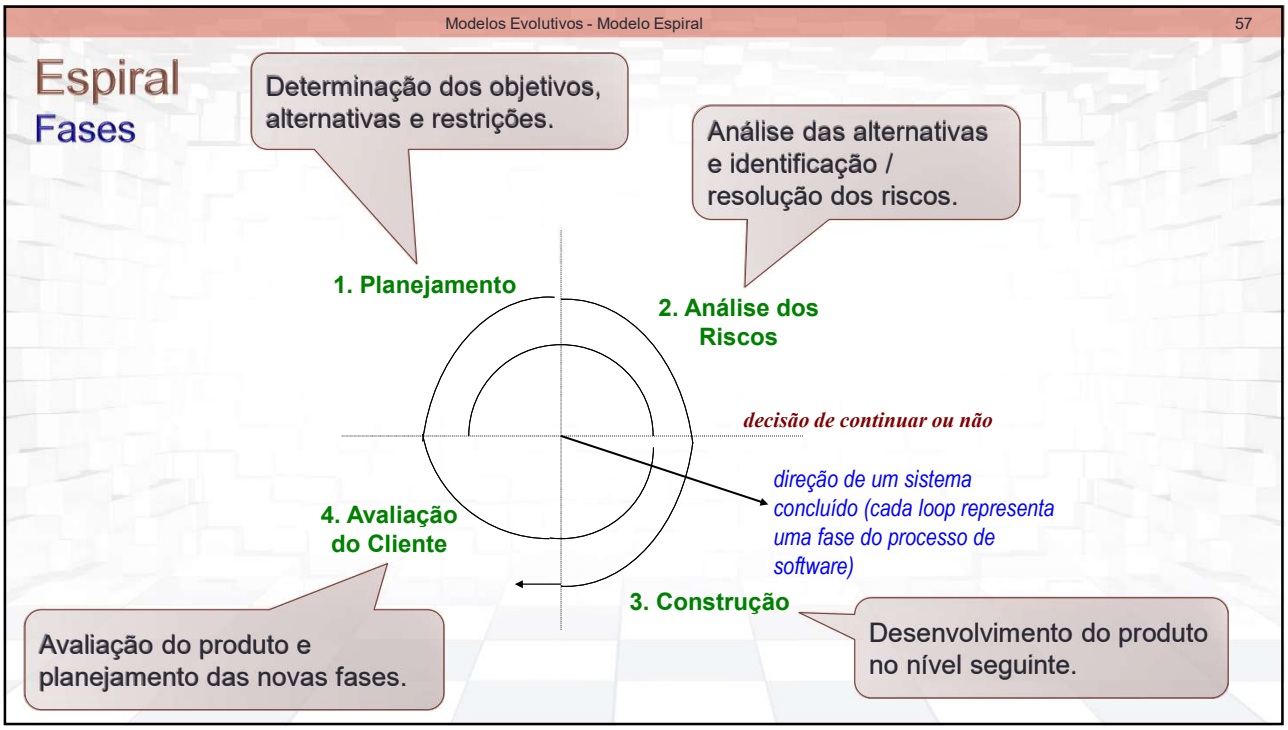
55

Espiral

- Engloba as melhores características dos modelos Cascata e Prototipação, adicionando um novo elemento: a Análise de Risco
- Segue a abordagem de passos sistemáticos do Ciclo de Vida Clássico (Cascata) incorporando-os numa estrutura iterativa que reflete mais realisticamente o mundo real
- Usa a Prototipação em qualquer etapa da evolução do produto como mecanismo de redução de riscos

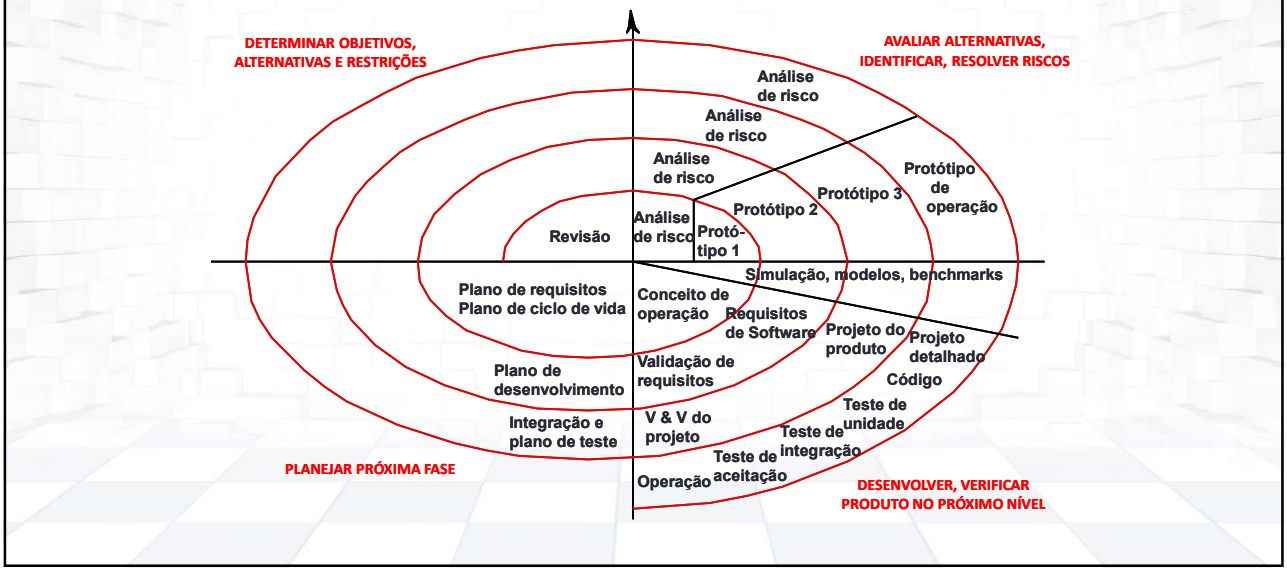
56

Espiral
Fases



57

Espiral
Exemplo de Fases



58

Espiral
Considerações Finais

- É uma abordagem bastante realista para o desenvolvimento de software em grande escala.
- Usa uma abordagem que capacita o desenvolvedor e o cliente a entender e reagir aos riscos em cada etapa evolutiva.
- Pode ser difícil convencer os clientes que uma abordagem "evolutiva" é controlável.
- Exige considerável experiência na determinação de riscos e depende dessa experiência para ter sucesso.

59

Técnicas de 4ª Geração

- **4GL (4th Generation Language)**: Concentra-se na capacidade de se especificar o software em um nível que esteja próximo à linguagem natural
- Engloba um conjunto de ferramentas de software que possibilitam que:
  - O sistema seja especificado em uma linguagem de alto nível
  - O código fonte seja gerado automaticamente a partir dessas especificações
  - https://pt.wikipedia.org/wiki/Linguagem_de_programação_de_quarta_geração
  - https://coggle.it/diagram/WPa0QDNf_QABL1UE/t/15-gerações-das-linguagens-de-programação

60

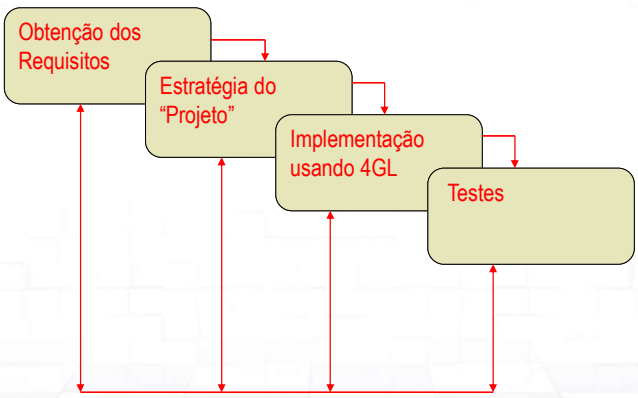
Técnicas de 4ª Geração
Ferramentas do Ambiente de Desenvolvimento

- O ambiente de desenvolvimento de software inclui as seguintes ferramentas:
  - Linguagens não procedimentais para consulta de banco de dados
  - Geração de relatórios
  - Manipulação de dados
  - Interação e definição de telas
  - Geração de códigos
  - Capacidade gráfica de alto nível
  - Capacidade de planilhas eletrônicas

Linguagens de alto nível: definem o que deve ser feito, mas não como.

61

Técnicas de 4ª Geração
Fases



62

Técnicas de 4ª Geração
Obtenção dos Requisitos

- O cliente descreve os requisitos os quais são traduzidos para um protótipo operacional
- O cliente pode estar inseguro quanto aos requisitos
- O cliente pode ser incapaz de especificar as informações de uma maneira que uma ferramenta 4GL as possa consumir
- As 4GLs atuais não são sofisticadas suficientemente para acomodar a verdadeira "linguagem natural"

63

Técnicas de 4ª Geração
Estratégia do Projeto

- Para pequenas aplicações é possível mover-se do passo de Obtenção dos Requisitos para o passo de Implementação usando uma linguagem de quarta geração
- Para grandes projetos é necessário desenvolver uma estratégia de projeto
  - De outra maneira ocorrerão os mesmos problemas encontrados quando se usa abordagem convencional
    - baixa qualidade

64

Técnicas de 4ª Geração
Implementação Usando 4GL

- Os resultados desejados são representados de maneira que haja geração automática de código
- Deve existir uma estrutura de dados com informações relevantes e que seja acessível pela 4GL

65

Técnicas de 4ª Geração
Testes

- 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

66

Técnicas de 4ª Geração
Considerações Finais

- **Proponentes**:
  - Redução dramática no tempo de desenvolvimento do software (aumento de produtividade)
- **Oponentes**:
  - As 4GL atuais não são mais fáceis de usar do que as linguagens de outras gerações
  - O código fonte produzido é ineficiente
  - A manutenibilidade de sistemas usando técnicas 4G ainda é questionável