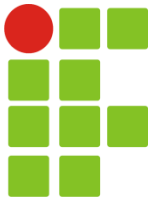


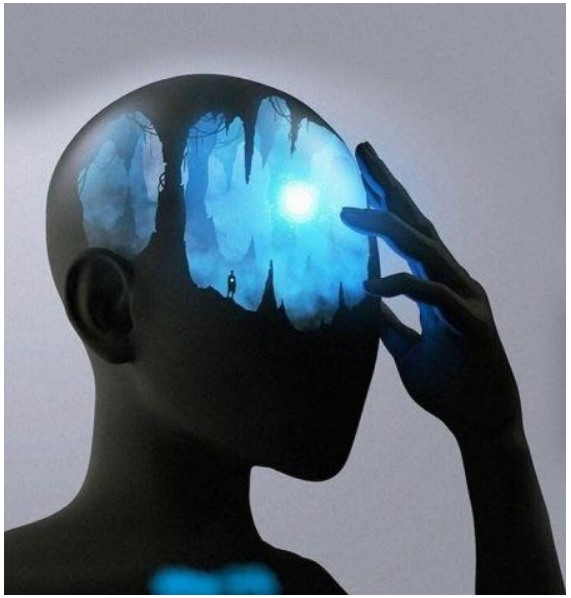
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO RIO GRANDE DO SUL
CAMPUS Canoas

ENGENHARIA DE SOFTWARE I

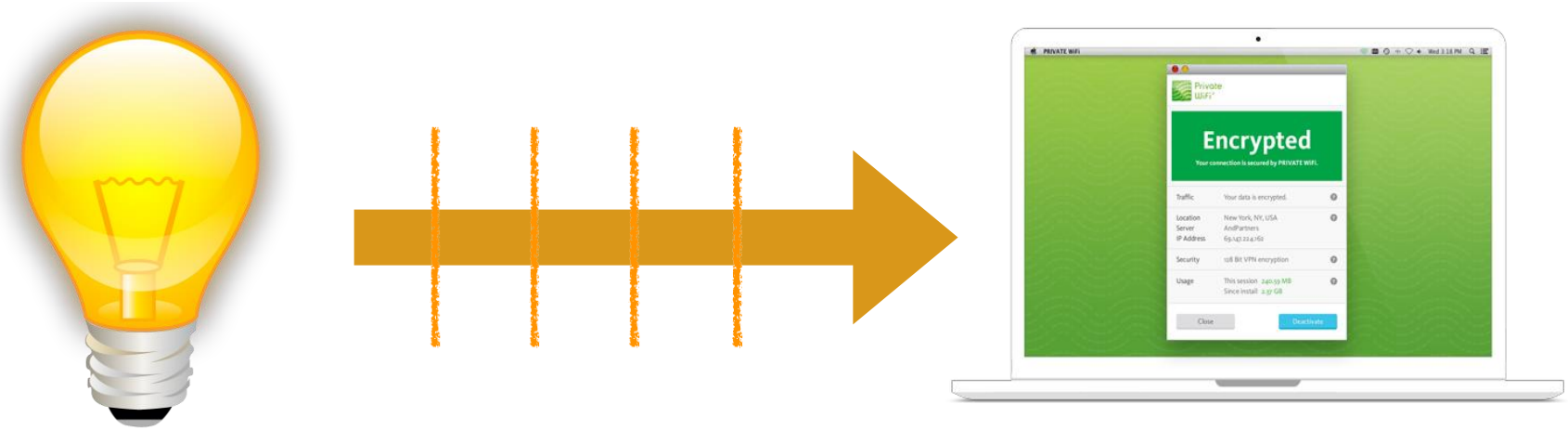
Aula 3

Mariano Nicolao
mariano.nicolao@canoas.ifrs.edu.br

Revisando e trazendo novos conceitos ....



Desenvolvimento de Software

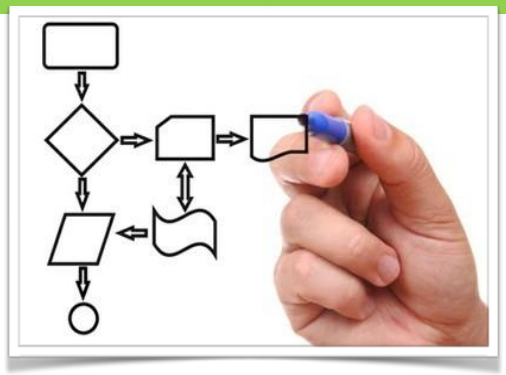


Processo de Software:

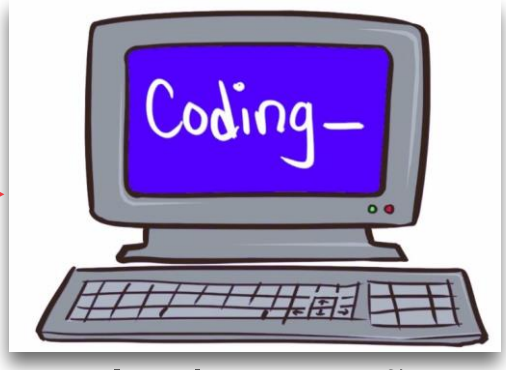
Quebrar o desenvolvimento em passos menores através de um processo sistemático e formal.

Fases do Desenvolvimento de Software

Engenharia de Requisitos



Análise e Projeto



Implementação

Verificação eValidação

Manutenção

Engenharia de Requisitos

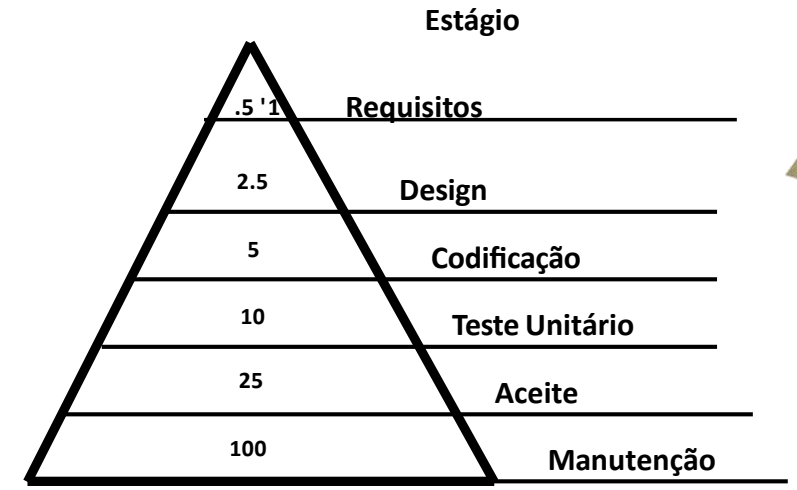
É o processo de definir as necessidades dos stakeholders que devem ser atendidas pelo software

Engenharia de Requisitos

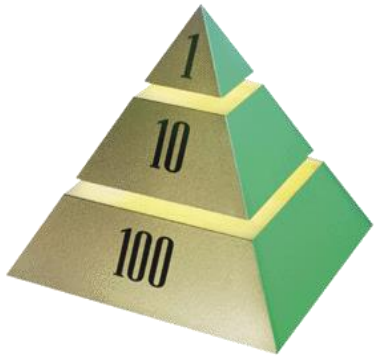
Esta fase é tão importante devido ao

custo da correção

A Regra 1-10-100



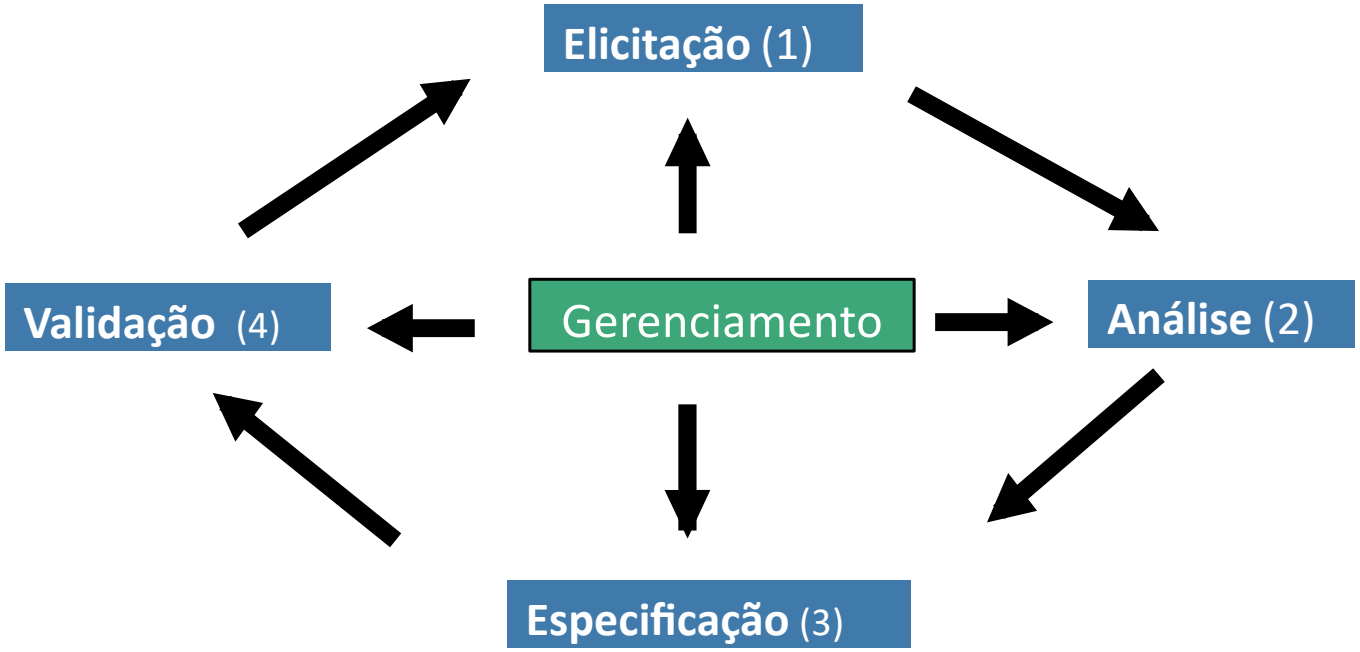
Boehm 1988

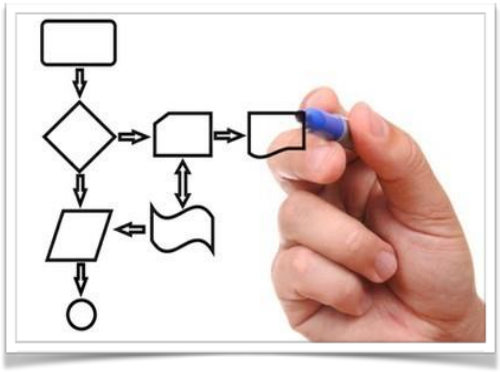


Custo relativo para corrigir erros: Quando introduzidos vs. quando corrigidos

Taxa média do custo 14:1 Grady 1989

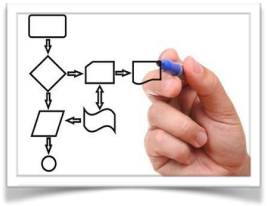
Engenharia de Requisitos





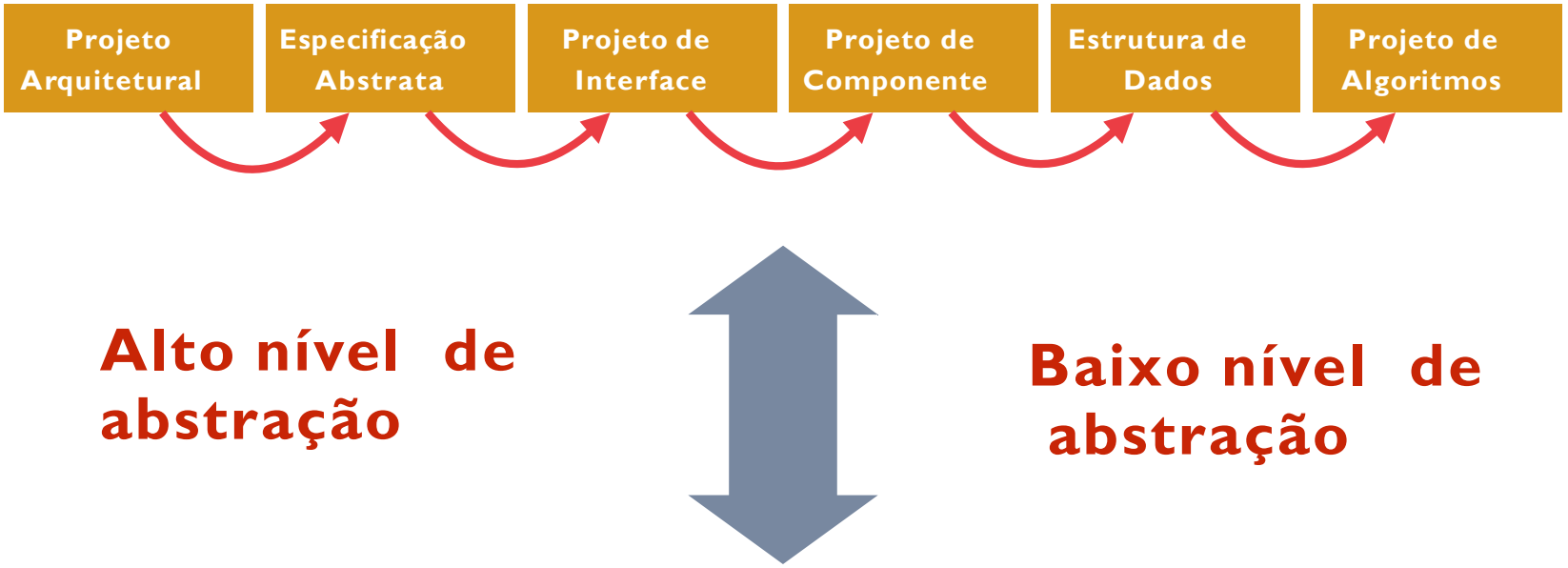
Análise e Projeto

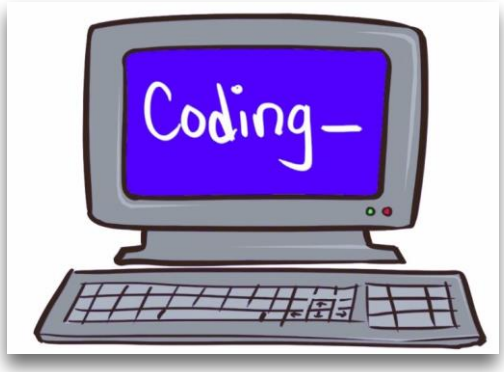
É o processo no qual os requisitos são analisados para produzir uma descrição da estrutura interna e da organização do sistema, servindo como base para sua construção.



Análise e Projeto

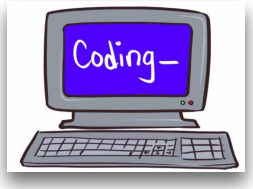
Atividades de Análise e Projeto





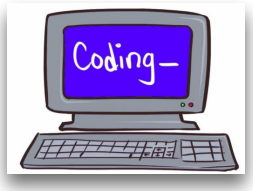
Implementação

É o processo de realizar o modelo gerado pela análise e projeto e efetivamente criando o sistema software.



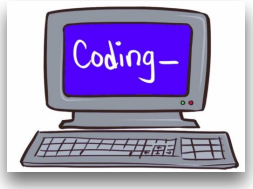
Implementação

- Há **alguns princípios** que afetam a forma que o software é construído:
  - **KISS ("keep it simple, stupid"):** Quando você pensou que simplificou o suficiente, simplifique mais um pouco - feature creep é inevitável, então comece pequeno."Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away." Antoine de Saint- Exupery.
  - **DRY (don't repeat yourself):** evite sempre duplicação de dados e de lógica (WET - write everything twice). Uma forma de avaliar isso é pensar em quantas partes do software é necessário mudanças devido a alteração de uma funcionalidade.
  - **Open/Close:** (principalmente para APIs) quando se está escrevendo código, deve-se fazer com que o mesmo seja **aberto a extensão e fechado a modificação,** separando assim o comportamento central (core) do adaptável. Quando se realiza um alteração no código, deve-se garantir a compatibilidade e estabilidade de futuras versões.



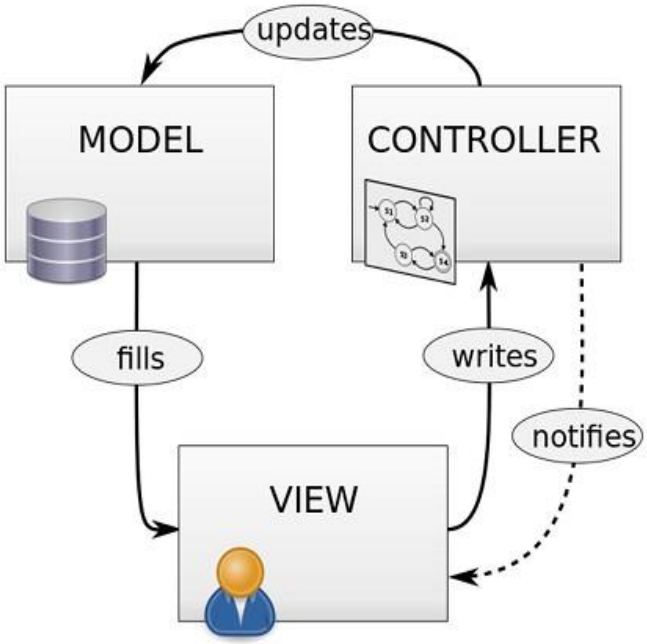
Implementação

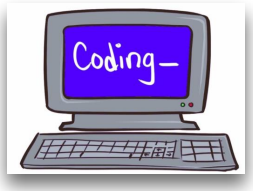
- Há **alguns princípios** que afetam a forma que o software é construído:
  - **Composição sobre Herança:** objetos com comportamentos complexos devem conter instâncias que implementam esses comportamentos ao invés de herdar de uma classe que contém esse comportamento (adicionando novos). A herança pode gerar confusão para o entendimento do código devido ao polimorfismo. A composição torna o código mais limpo, flexível e fácil de manter pois cada comportamento complexo é implementado em classe individual.
  - **Responsabilidade Única/Single Responsibility:** cada classe/módulo deve se preocupar em fornecer uma pequena parte de uma funcionalidade específica: "A class should have only one reason to change." (Robert C. Martin). Classes geralmente começam assim, mas podem facilmente evoluir para uma Classe Deus de milhares de linhas de código. Nesse ponto, deve-se quebra-las em classes menores.



Implementação

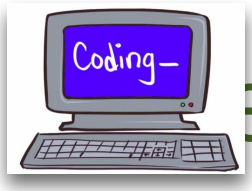
- Há **alguns princípios** que afetam a forma que o software é construído:
  - **Separação de Conceitos/Separation of Concerns:** é semelhante a anterior, mas mais abstrata. O programa deve ser projetado para não ter sobreposição de encapsulamento e cada encapsulamento não saber da existência de outro. Por exemplo:
    - o código que gerencia a persistência não sabe como renderizar/apresentar o dado na Web.
    - quando o usuário passa o dado para camada de visão (renderização), esta passa passa para a camada de negócio (processamento), que passa para a camada de persistência (dados). Cada parte/camada tem sua responsabilidade





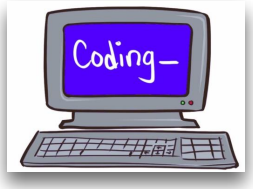
Implementação

- Há **alguns princípios** que afetam a forma que o software é construído:
  - **YAGNI ("you aren't gonna need it"):** você nunca deve codificar uma funcionalidade que você irá precisar no futuro (provavelmente você não irá precisar e isso é uma perda de tempo que irá aumentar a complexidade do código). Isso é uma aplicação do princípio KISS e uma resposta aqueles que tentam aplicar o DRY muito a sério (a dica é usar o DRY com parcimônia).
  - **Evite otimização prematura:** é um princípio similar ao YAGNI. O YAGNI endereça a tendência de implementar comportamentos antes dos mesmos serem necessários enquanto esse de otimizar algoritmos antes que isso seja necessário/ solicitado.



ementação

- Há **alguns princípios** que afetam a forma que o software é construído:
  - **Refatore, refatore, refatore:** uma das mais difíceis verdades de um programador inexperiente aceitar é que o **código raramente fica certo da primeira vez**. Quando se implementa uma funcionalidade e posteriormente sua complexidade aumenta, pode-se ficar difícil entender no futuro como aquela funcionalidade foi implementada. Lembre-se do lema dos escoteiros: "Deixe o acampamento mais limpo que o encontrou."
    - "Any fool can write code that a computer can understand. Good programmers write code that humans can understand."
      — Martin Fowler, Refactoring: Improving the Design of Existing Code



Implementação

- Há **alguns princípios** que afetam a forma que o software é construído:
  - **Clean Code ao invés de Clever Code:** deixe seu ego na porta e
    - evite escrever código complexo/clever - aquele que parece mais um enigma do que uma solução. A verdade é que ninguém se importa com seu código se ele não for limpo e compreensível.
    - Por exemplo, empacotar muita lógica em apenas uma linha,
    - explorar as minúcias da linguagem para fazer comandos estranhos mas funcionais, enfim, tudo que alguém possa falar "Espera, o que ele quis fazer com esse código?'. Veja algumas dicas em: https: cleaner-better-code/

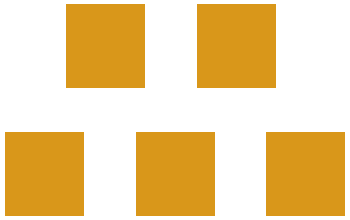
Verificação e Validação

É o processo de avaliar se o software atende a sua especificação e a sua razão de existir.

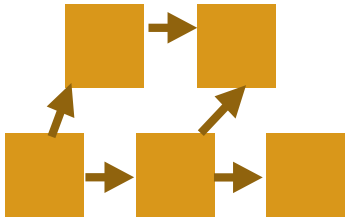
Verificação e Validação

Validação: estamos construindo **o sistema certo**?

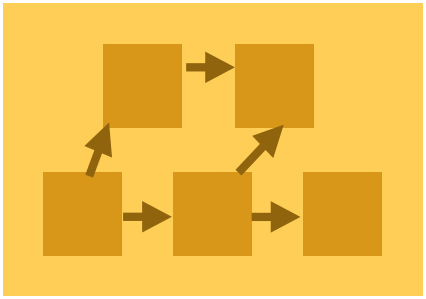
Verificação: estamos construindo **certo o sistema**?



Unidade



Integração



Sistema

Manutenção

É o processo de manter o software útil enquanto ele evolui ao longo de seu ciclo de vida.

Manutenção

- Quando o software está em operação, muita coisa pode acontecer: mudanças de ambiente, novas necessidades dos usuários, correção de bugs,etc.

- Basicamente temos três tipos de manutenções:
  - **Manutenções Corretivas**: eliminar falhas encontradas em produção.
  - **Manutenções Evolutivas**: visam agregar novas funcionalidades e melhorias no sistema
  - **Manutenções Adaptativas**: adaptar o software a nova realidade ou novo ambiente externo, normalmente imposto (leis, regras, etc.).

- Sempre que aplicar uma manutenção, deve-se realizar **teste de regressão** para verificar se ela não introduziu nenhuma nova falha.

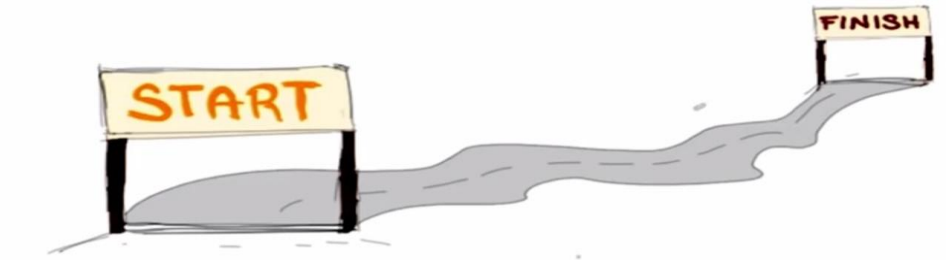
Questões

Quais são as tradicionais fases do desenvolvimento de software?

[ ] Engenharia de Requisitos, Análise e Projeto, Abstração, Implementação, Verificação e Validação

[ ] Análise e Projeto, Otimização, Implementação, Verificação e Validação, Manutenção

[ ] Engenharia de Requisitos, Análise e Projeto, Implementação, Verificação e Validação, Manutenção



Modelo de Processo de Desenvolvimento

Modelo definido de tudo que deve acontecer desde o início até o final de um processo de desenvolvimento de software

Modelo de Processo de Desenvolvimento

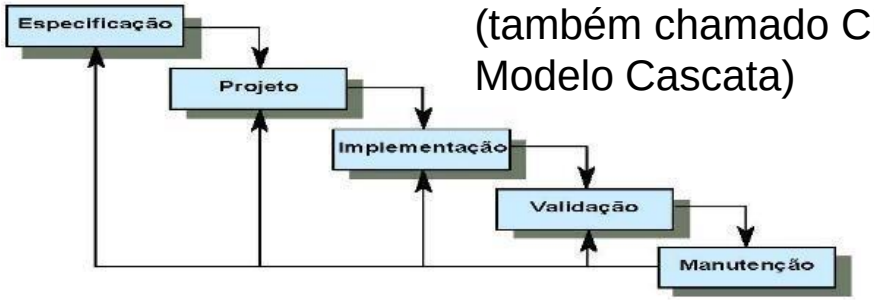
- O objetivo é determinar a **ordem** em que cada atividade é executada

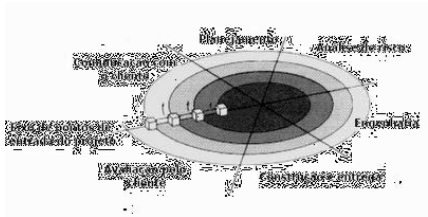
- além do critério de transição

- Basicamente o modelo deve dizer: (1) o que nós devemos fazer e (2) quanto tempo nós devemos continuar executando cada atividade.

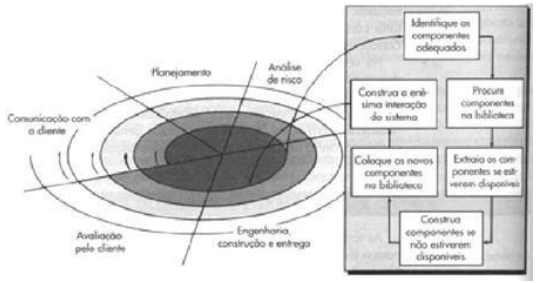
Modelos de Processo de Software ou Processos de Desenvolvimento de Software

O Modelo Seqüencial Linear
(também chamado Ciclo de Vida Clássico ou Modelo Cascata)



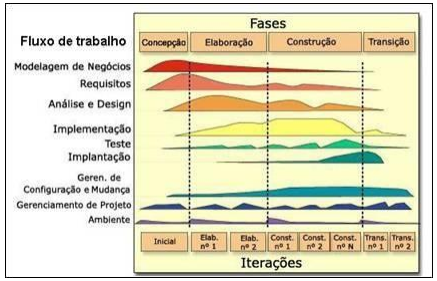


O Modelo Espiral



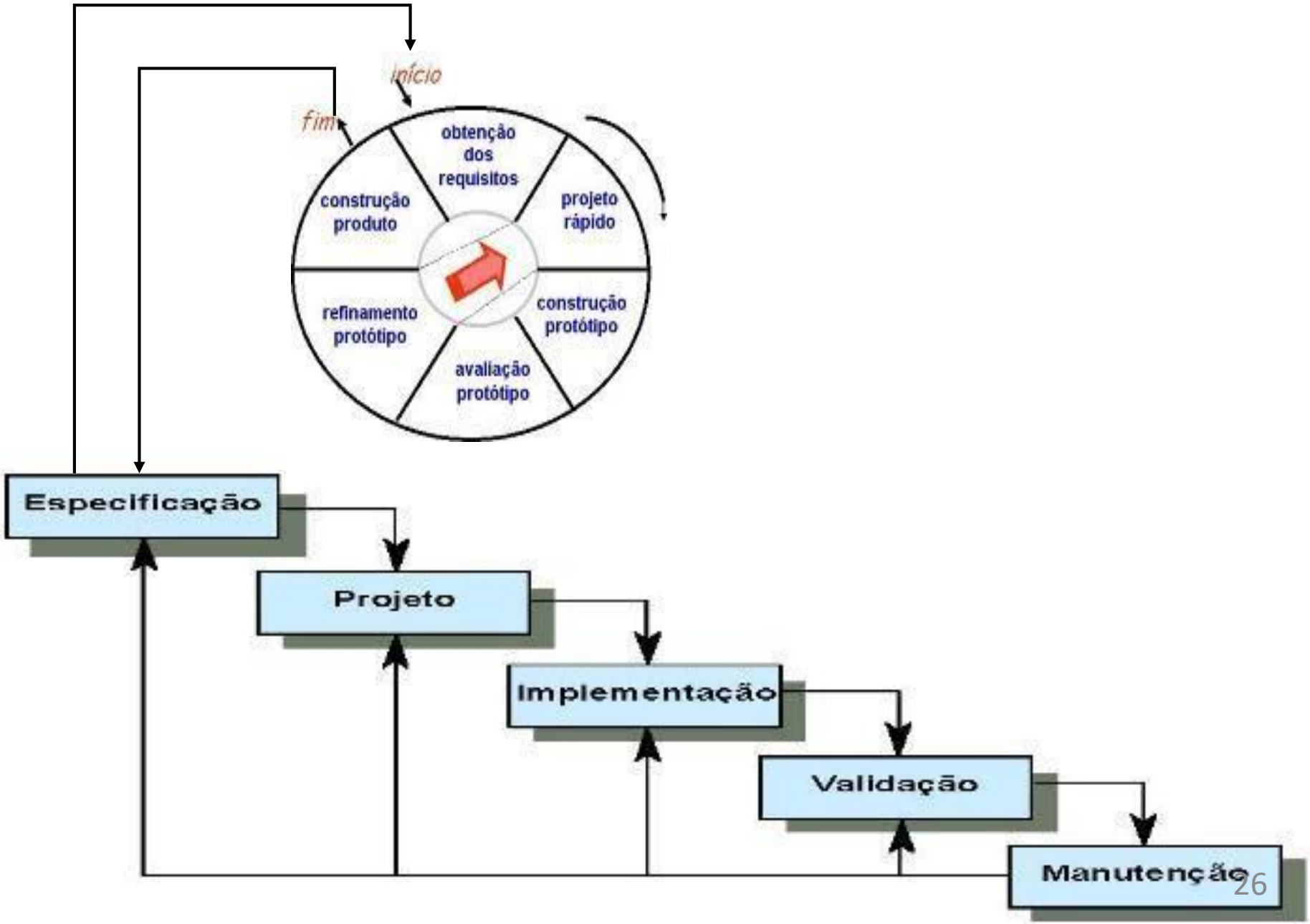
O Modelo Baseado em Componentes

O Paradigma de Prototipação



Processo Unificado

Prototipação


início
fim
obtenção dos requisitos
projeto rápido
construção protótipo
avaliação protótipo
refinamento protótipo
construção produto
Especificação
Projeto
Implementação
Validação
Manutenção

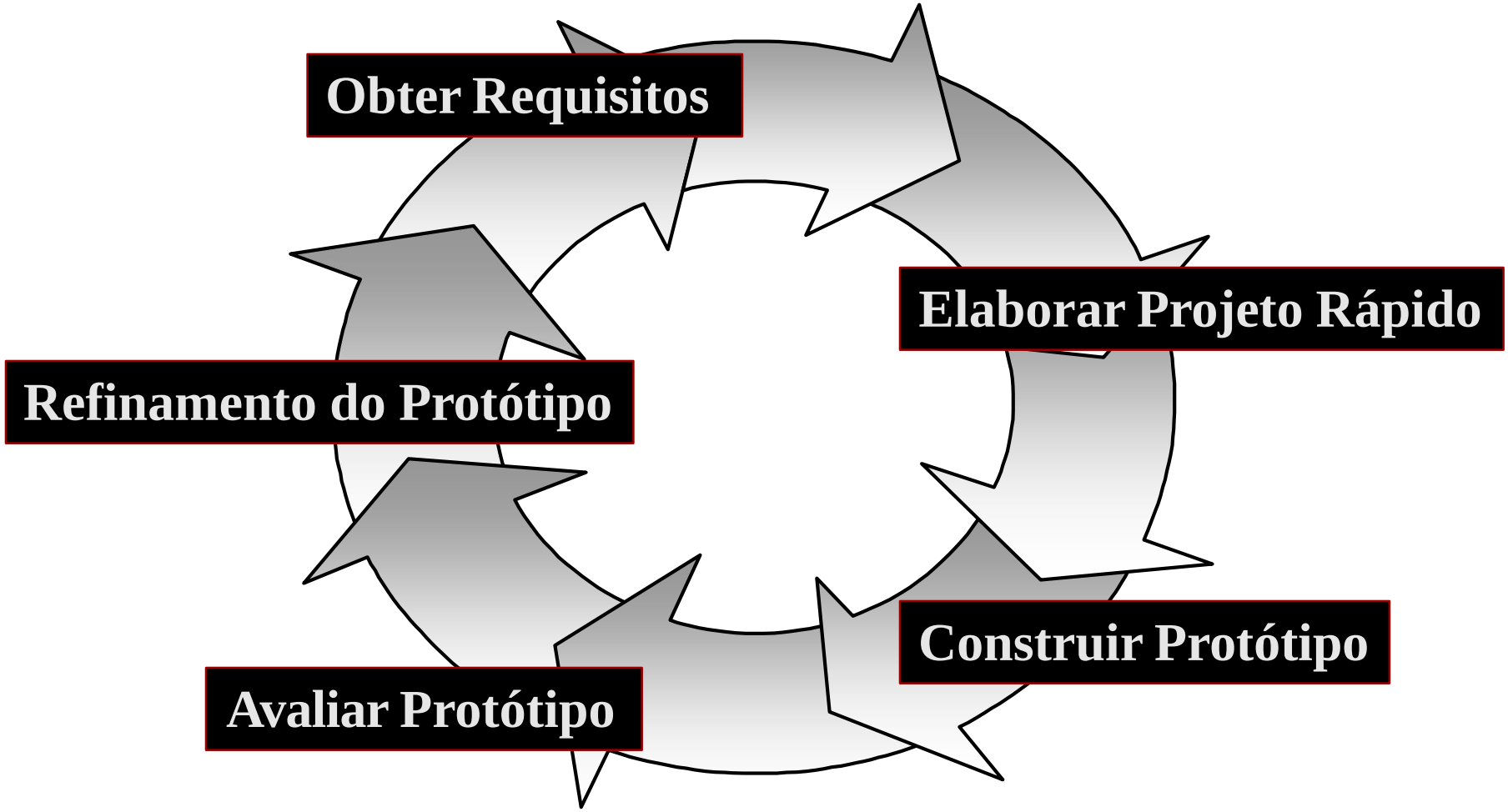
O Modelo de Prototipação

- o objetivo é entender os requisitos do usuário e, assim, obter uma melhor definição dos requisitos do sistema.

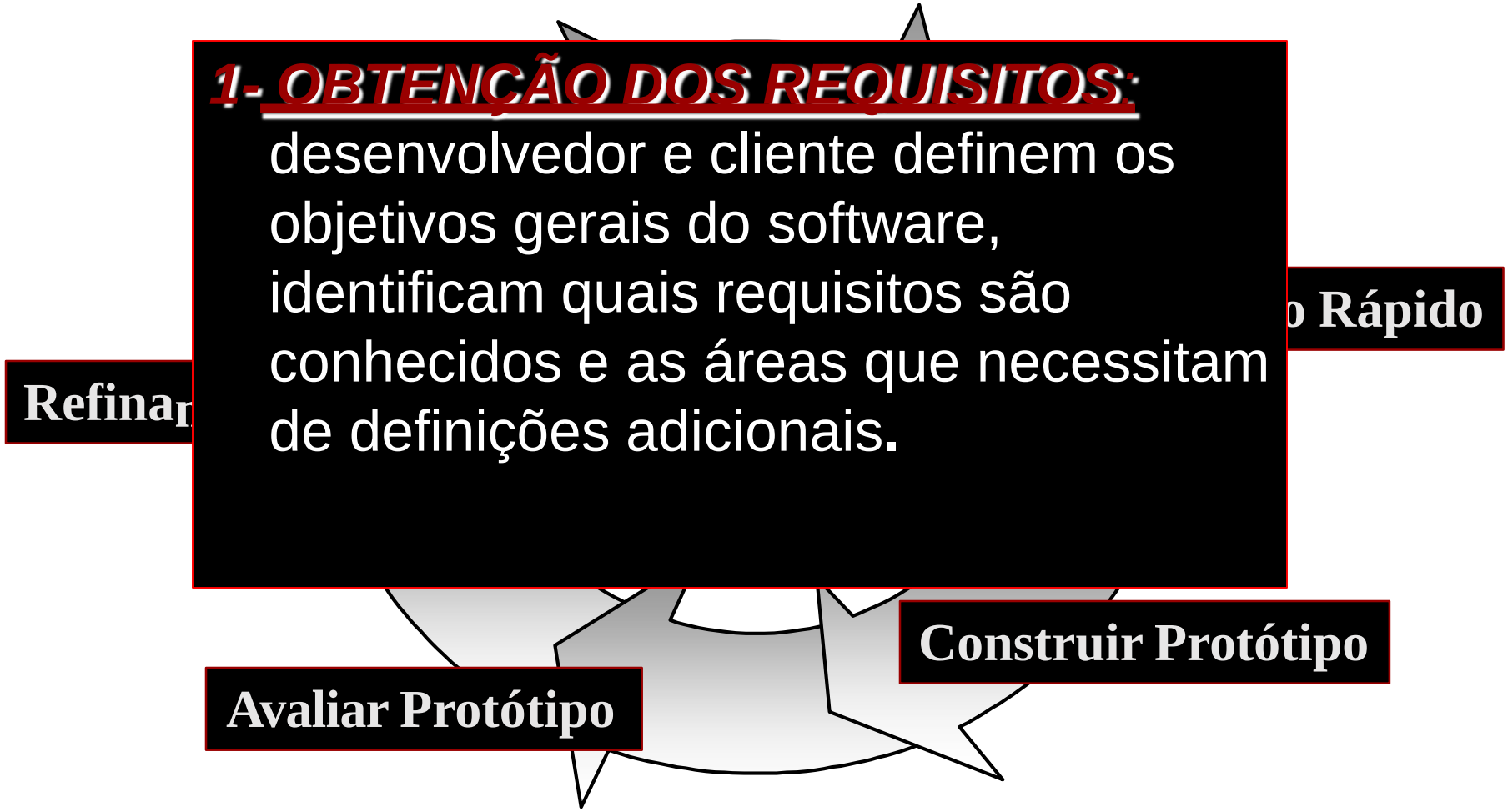
- possibilita que o desenvolvedor crie um modelo (protótipo)do software que deve ser construído

- apropriado para quando o cliente não definiu detalhadamente os requisitos.

O Paradigma de Prototipação para obtenção dos requisitos



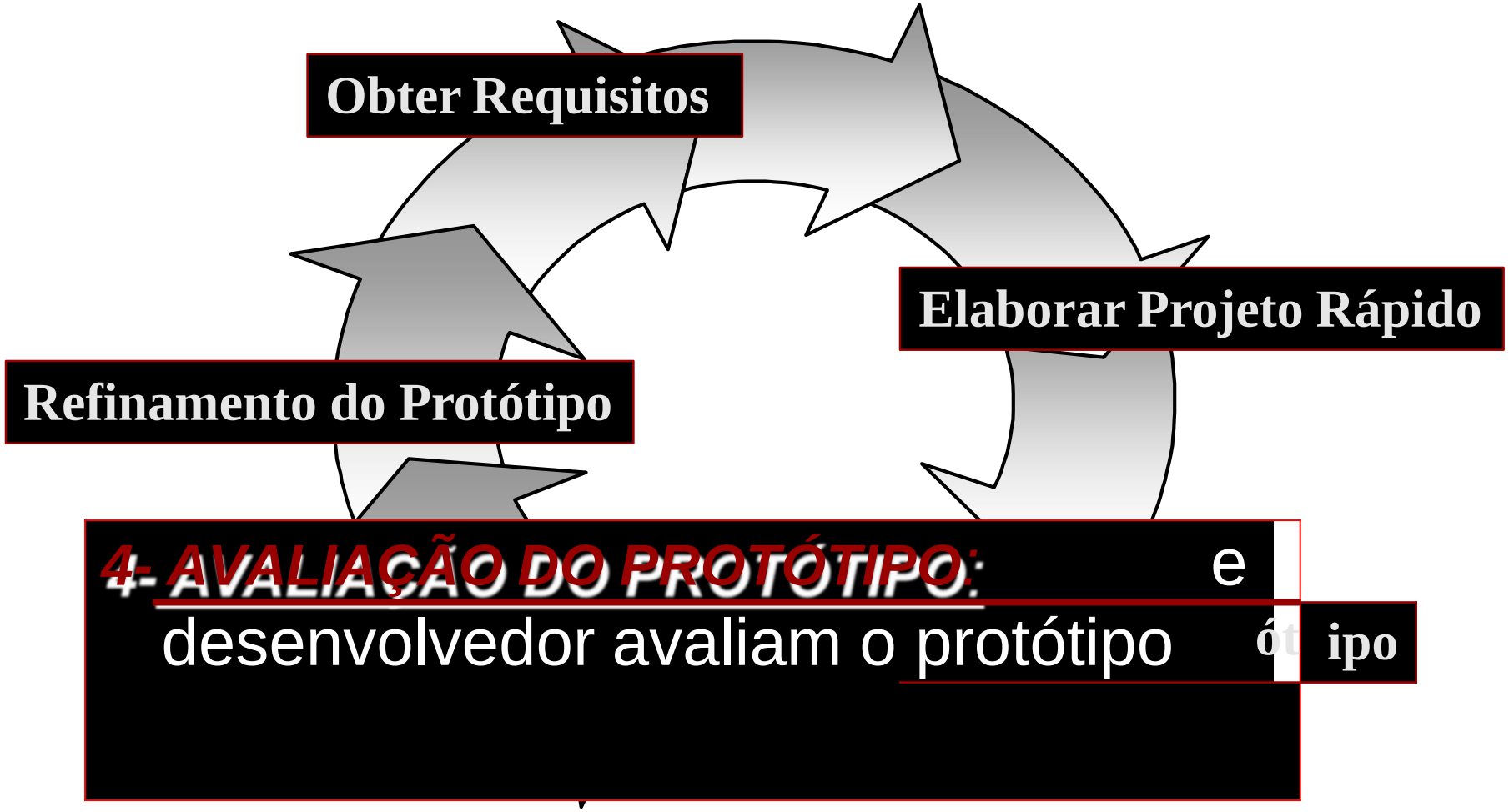
O Paradigma de Prototipação para obtenção dos requisitos



O Paradigma de Prototipação para obtenção dos requisitos

O Paradigma de Prototipação para obtenção dos requisitos

O Paradigma de Prototipação para obtenção dos requisitos

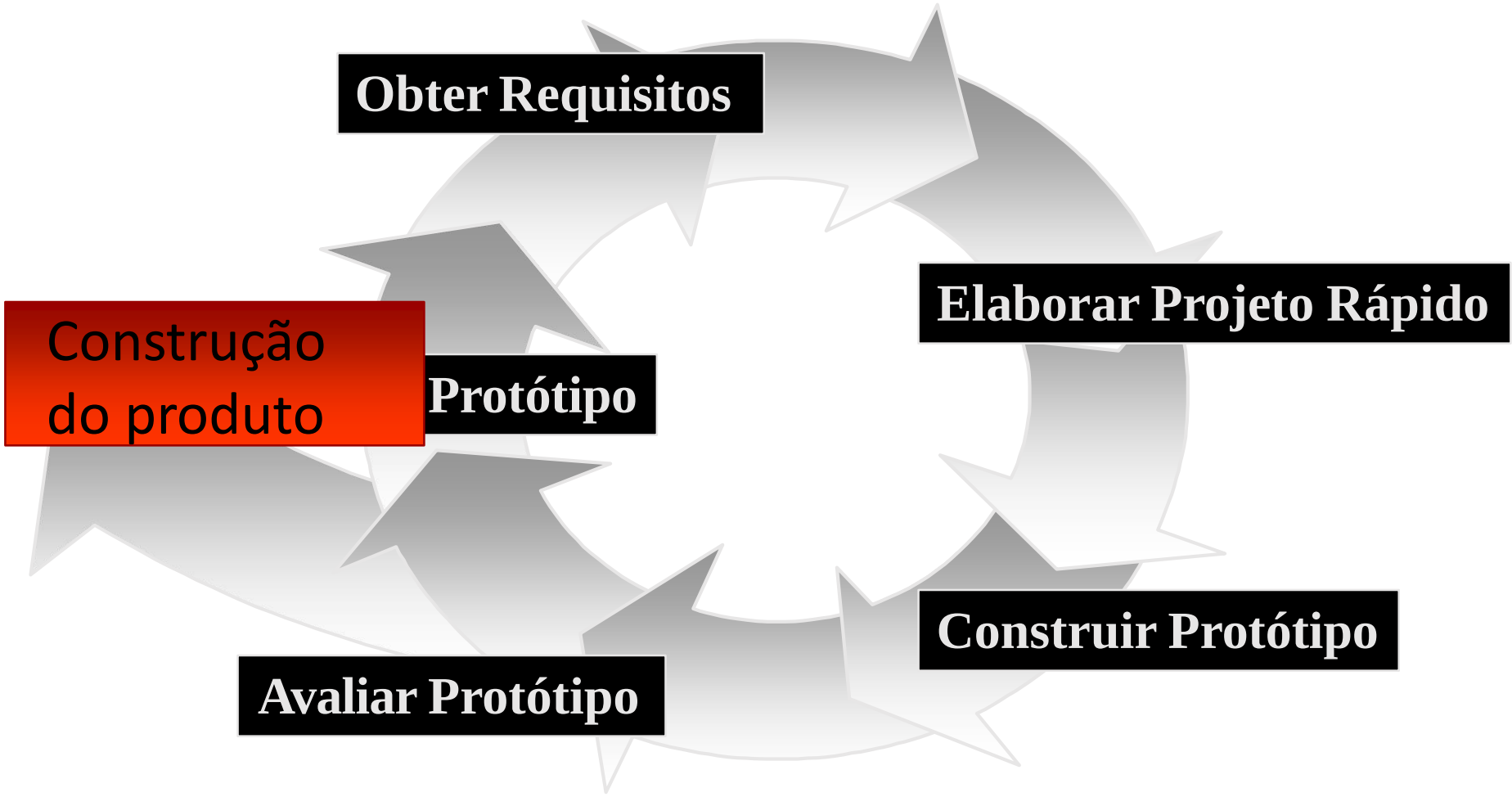


O Paradigma de Prototipação para obtenção dos requisitos

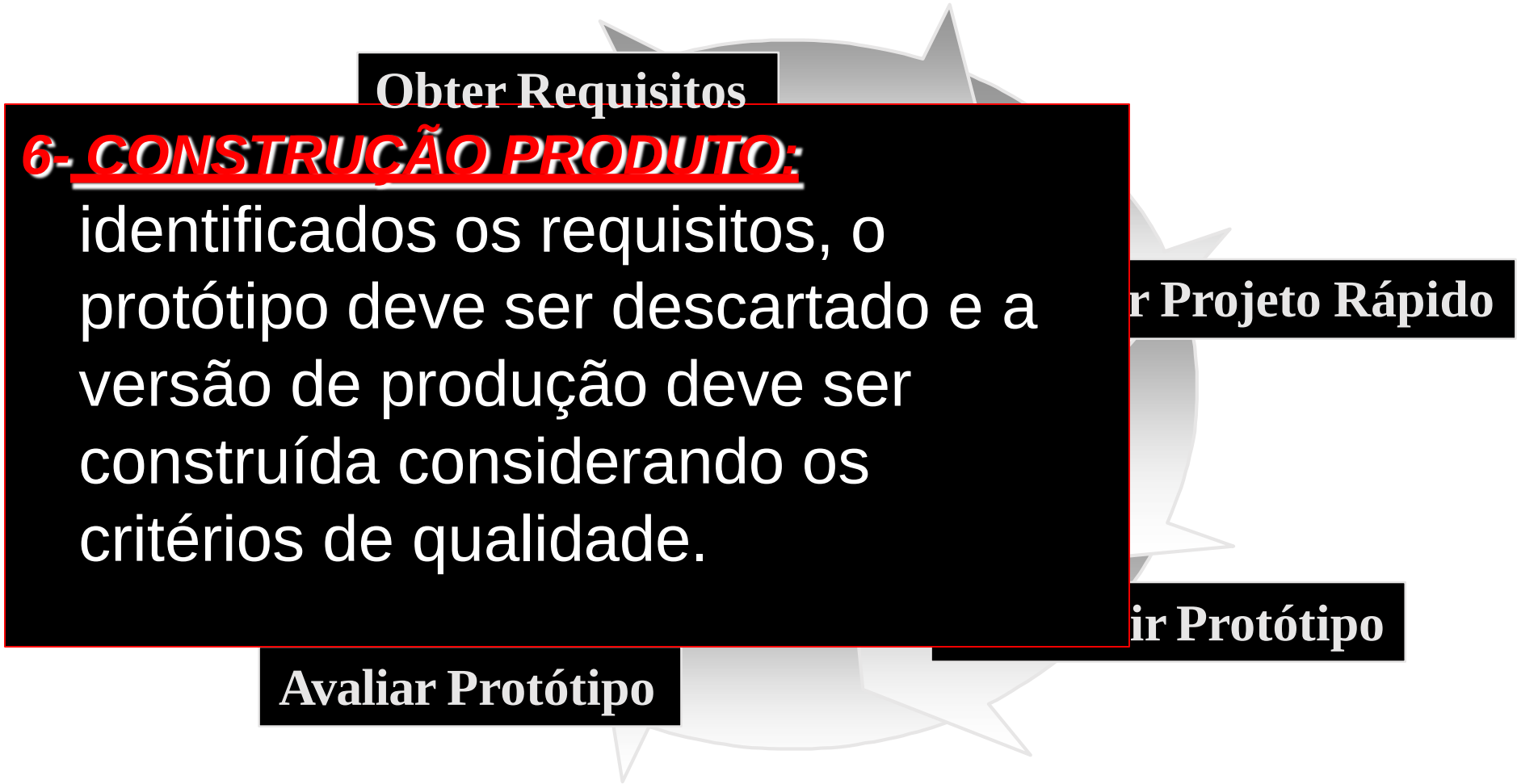
5- REFINAMENTO DO PROTÓTIPO:

desenvolvedor refinam os requisitos do software a ser desenvolvido.

O Paradigma de Prototipação para obtenção dos requisitos

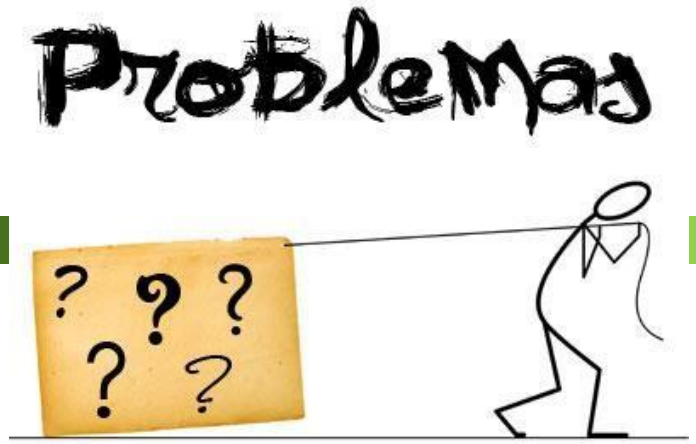


O Paradigma de Prototipação para obtenção dos requisitos



6- CONSTRUÇÃO PRODUTO:

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



Problemas com a Prototipação

- cliente não sabe que o software que ele vê não considerou, durante o desenvolvimento, a qualidade global e a mantenabilidade a longo prazo;

- desenvolvedor frequentemente faz uma implementação comprometida (utilizando o que está disponível) com o objetivo de produzir rapidamente um protótipo.

Comentários sobre o Paradigma de Prototipação

- Ainda que possam ocorrer problemas, a prototipação é um ciclo de vida eficiente.
- A chave é definir as regras do jogo logo no começo.
- O cliente e o desenvolvedor devem ambos concordar que o protótipo seja construído para servir como um mecanismo para definir os requisitos.

Ferramentas:

- https://marvelapp.com/

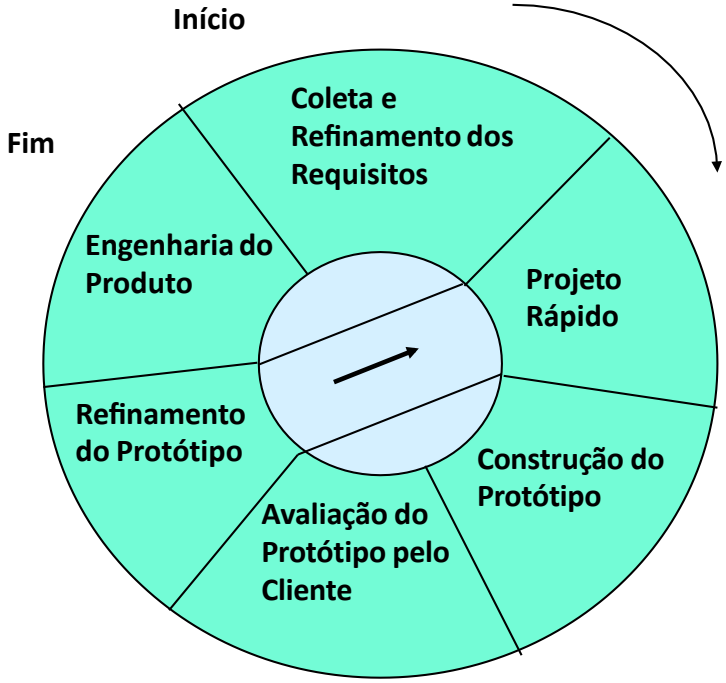
Prototipação Evolutiva

Um modelo no qual nem todos os requisitos precisam ser entendidos...

Prototipação Evolutiva

Conceito Inicial	Projeto e Implementação Inicial do Protótipo	Refinar o protótipo até ser aceitável	Completar e lançar o protótipo

- O sistema é continuamente refeito e reconstruído, portanto ideal quando nem todos os requisitos estão compreendidos
- O desenvolvimento se dá das partes do sistema que os desenvolvedores compreendem, ao invés de desenvolver todo o sistema.
  - O sistema é entregue para o usuário, avaliado e depois segue para a nova iteração, na qual o protótipo é melhorado ou incrementado.

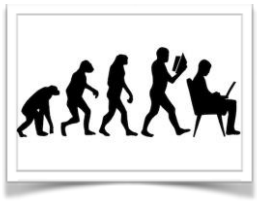


Prototipação Evolutiva

- **Vantagens:**
  - Feedback imediato
  - Prova de conceito.
  - Visibilidade antecipada do produto.
  - Encoraja participação dos usuários.
  - Exercício para identificar inconsistências de análise e projeto.
  - Redução do esforço no desenvolvimento.
  - Refina potenciais riscos de projeto.

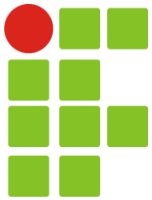
Prototipação Evolutiva

- **Desvantagens:**
  - É muito difícil de planejar quantos ciclos de desenvolvimento vai levar
  - Atividade de análise simplificada.
  - Confusão do usuário entre protótipo e o sistema real (performance, validação, etc).
  - Tempo e custo excessivo para desenvolvimento.
  - Falta de visibilidade do processo.
  - Dificuldade em manter a integridade entre protótipo e o software desenvolvido durante sua manutenção.



Prototipação Evolutiva

- **Tipos/Classificações de Prototipação**
  - **Vertical**: funcionalidade em profundidade;
  - **Horizontal**: amplitude de funções superficiais.
  - **Evolucionário**: evoluir até chegar a um produto;
  - **Descartável**: entender requisitos.



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E
TECNOLOGIA DO RIO GRANDE DO SUL
CAMPUS Canoas

RATIONAL UNIFIED PROCESS - RUP

Introdução

- Modelo de processo de software proprietário.
- Desenvolvido pela empresa Rational Software Corporation.
- Em 2003 a empresa foi adquirida pela IBM. Então o RUP ganhou uma novo nome: IRUP.
- Utiliza a abordagem da orientação a objetos.
- Projetado e documentado utilizando notações da UML.
- Utiliza práticas e técnicas aprovadas no mercado.

Introdução

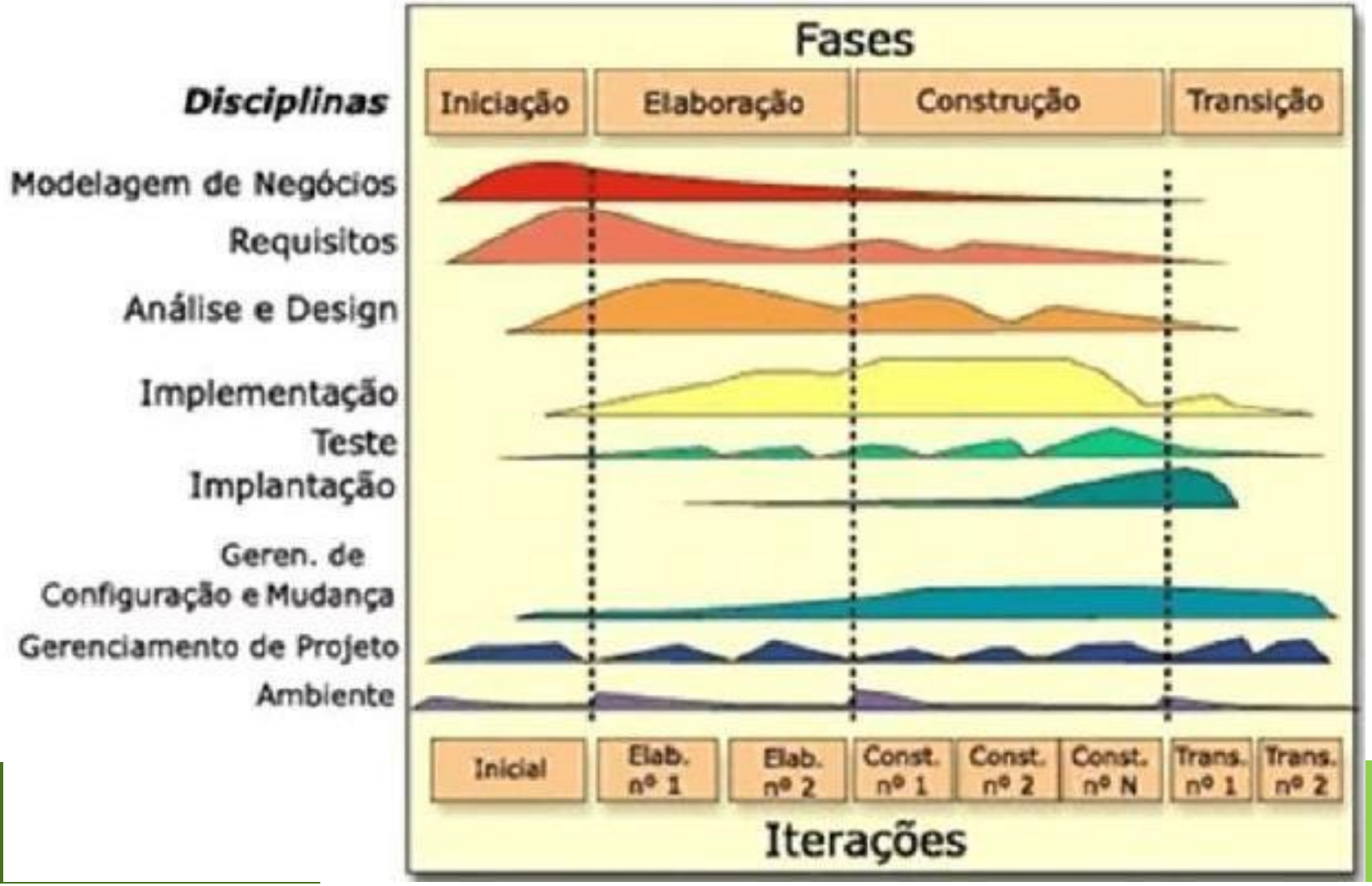
- Está muito próximo da Engenharia de Software baseada em componentes.
- Possui sua preocupação baseada no atendimento do negócio.
- Possui um conceito de processo de software híbrido.
- Traz elementos de todos os modelos de processo de software.
- Baseado em iterações.
- Ilustra boas práticas de especificação e projeto.

RUP

- Descrito a partir de 3 perspectivas:

- Dinâmica: são as fases do modelo ao longo do tempo.
- Estática: são as atividades realizadas no processo.
- Prática: boas práticas a serem utilizadas durante o processo.

RUP


Fases
Disciplinas
Iniciação
Elaboração
Construção
Transição
Modelagem de Negócios
Requisitos
Análise e Design
Implementação
Teste
Implantação
Geren. de Configuração e Mudança
Gerenciamento de Projeto
Ambiente
Inicial
Elab. nº 1
Elab. nº 2
Const. nº 1
Const. nº 2
Const. nº N
Trans. nº 1
Trans. nº 2
Iterações

Fases do RUP (Dinâmica)

- O modelo é constituído por 4 fases:
  - Concepção
  - Elaboração
  - Construção
  - Transição

Fases do RUP (Dinâmica)

- **Concepção:** estabelecer um business case para o sistema.
- Avaliação do ambiente de negócio em relação à contribuição de um sistema para o negócio (**ESCOPO**).
- Identificar também as entidades externas: pessoas e sistemas, que irão interagir com o sistema.
- Depois utilizamos essas informações para avaliar a importância do sistema com o negócio.

Fases do RUP (Dinâmica)

- Os objetivos da fase **Elaboração** são:
  - Desenvolver um entendimento do domínio do problema;
  - Estabelecer um framework de arquitetura para o sistema;
  - Desenvolver o plano de projeto;
  - Identificar os riscos do projeto.
- Planejamento.

Fases do RUP (Dinâmica)

- **Construção:** está diretamente ligada ao projeto, programação e testes. (**DESENVOLVIMENTO**)
- Partes do sistema são implementadas paralelamente e integradas durante essa fase.
- O resultado dessa fase é um software funcional e toda sua documentação pronta para ser liberada para os usuários.

Fases do RUP (Dinâmica)

- **Transição:** é a última fase do RUP.
- Transferência do desenvolvimento para o usuário.
- Entrada do sistema em produção.
- Fase onerosa e problemática.
- **Implantação**

Disciplinas ou *Workflows* (Estática)

- Foca nas atividades que ocorrem durante o processo de desenvolvimento.
- Também conhecidas como *WORKFLOWS*.
- Todos os workflows podem ser ativados em todos os estágios do processo.
- Existem no total 9 *workflows*:
  - 6 workflows de processo
  - 3 workflows de apoio

Disciplinas ou Workflows (Estática)

- **Modelagem de negócio:** processo de negócio são modelados utilizando casos de uso.
- **Requisitos:** identificar os agentes que interagem com o sistema e os casos de uso são desenvolvidos.
- **Análise e projeto:** modelo de projeto é criado e documentado utilizando os modelos de arquitetura, modelos de componente, modelos de objeto e modelos de sequência.
- **Implementação:** componentes de sistema são implementados e estruturados em subsistemas de implementação.

Disciplinas ou Workflows (Estática)

- **Teste:** processo iterativo realizado em conjunto com a implementação.
- **Implantação:** versão final do produto é criada, distribuída aos usuários e instalada.
- **Gerenciamento de configuração e mudanças:** controla as mudanças do sistema.
- **Gerenciamento de projetos:** gerencia o desenvolvimento do sistema.
- **Ambiente:** relacionado à disponibilização de ferramentas apropriadas de software para a equipe de desenvolvimento.

Linhas Mestras (Prática)

- O RUP recomenda 6 práticas fundamentais (ou linhas mestras):
  1. Desenvolver o software de forma iterativa: incrementos de software priorizados e entregues.
  2. Gerenciar requisitos: documentação e acompanhamento das mudanças dos requisitos.
  3. Usar arquitetura baseada em componentes: estruturar a arquitetura de sistema de componentes.

Linhas Mestras (Prática)

4. Modelaro software graficamente: modelos gráficos da UML.
5. Verificar a qualidade do software: atender aos padrões de qualidade da organização.
6. Controlar mudanças do software: utilizando um SGM e procedimentos bem definidos.

Exercícios

1. Julgue o item que se segue a respeito da prototipação relacionada ao desenvolvimento de software.

- Entre as atividades de prototipação de software, está o desenvolvimento rápido de software para validar requisitos.

- ( ) Certo
- ( )Errado

Exercícios

2. Julgue o item que se segue a respeito da prototipação relacionada ao desenvolvimento de software.

A prototipação pode ser considerada como uma atividade que reduz riscos de desenvolvimento relacionados aos requisitos de um software.

- ( ) Certo
- ( )Errado

Exercícios

3. No que diz respeito aos processos da engenharia de requisitos de um sistema de informação, julgue o seguinte item.
Em um protótipo para validar os requisitos de um software, é admissível deixar de fora os requisitos não funcionais ou reduzir seus padrões.

( ) Certo
( )Errado

Exercícios

4. Ao analisar a aplicação da prototipação em seus projetos de software, decidiu-se utilizar um processo que define 4 etapas para o desenvolvimento de protótipos, sendo essas etapas:
I. Avaliar Protótipo. II. Estabelecer Objetivos do Protótipo. III. Desenvolver o Protótipo. IV. Definir a Funcionalidade do Protótipo.
A ordem sequencial correta para a execução dessas quatro etapas é

A) I, IV, II e III.
B) II, IV, III e I.
C) III, II, IV e I.
D) IV, I, II e III.
E) IV, II, I e III.

Exercícios

5. O sistema “SGRH” vem sendo desenvolvido com a utilização de prototipagem. O sistema tem prazo previsto de um ano para a entrega do produto final ao cliente. Porém, após analisar o protótipo, durante o primeiro mês de desenvolvimento, ainda no levantamento de requisitos, o cliente concluiu que esse prazo era muito extenso, visto que, em sua opinião, o desenvolvimento estava muito adiantado.

É uma desvantagem do modelo de prototipagem relacionado ao fato relatado acima:

A) estabelecer uma participação mais formal do cliente;
B) tornar informal o processo de modelagem de requisitos;
C) trazer uma falsa impressão sobre a complexidade do projeto;
D) ser incompatível com processos formais de desenvolvimento;
E) obedecer a um fluxo sequencial.

Exercícios

6. Pesquise/explique o que é RAD (Rapid Application Development.) e (JAD) Joint application development – Essas técnicas são normalmente utilizadas em quais Modelos ou Processos de Desenvolvimento de Software? Coloque a fonte de consulta.

Exercícios

7. Julgue a afirmação abaixo com V ou F

( ) - O processo unificado é um dos padrões mais importantes da indústria de *software* atual, caracterizando-se por ser dirigido por casos de uso, centrado na arquitetura, interativo e incremental, além de ter foco em riscos.

Exercícios

8. Considere os seguintes objetivos das Fases do Rational Unified Process - RUP:

I. Analisar de forma mais detalhada o domínio do problema, revisando os riscos que o projeto pode sofrer. A arquitetura do projeto inicia-se com sua forma básica elaborada. Indagações como "O plano do projeto é confiável?", "Os custos são admissíveis?" são esclarecidas nesta fase.
II. Abranger as tarefas de comunicação com o cliente e o planejamento. É feito um plano de projeto avaliando os possíveis riscos, as estimativas de custo e prazos, estabelecendo as prioridades, o levantamento dos requisitos do sistema e a análise preliminar. Nesta fase, deve haver concordância dos *stakeholders* quanto ao escopo do projeto.
III. Disponibilizar o sistema de forma que seja compreendido pelo usuário final. As atividades desta fase incluem o treinamento dos usuários finais e a realização de testes da versão beta do sistema visando garantir a sua qualidade.
IV. Desenvolver ou adquirir componentes de *software*. O principal objetivo desta fase é codificação do *softwar*e, com foco nos componentes e outros recursos do sistema.

Os objetivos I a IV estão correlacionados, correta e respectivamente, às fases de
A Concepção, Elaboração, Construção e Transição.
B Elaboração, Concepção, Construção e Transição.
C Transição, Elaboração, Concepção e Construção.
D Elaboração, Concepção, Transição e Construção.
E Concepção, Construção, Elaboração e Transição.

Exercícios

9. No Processo Unificado, um *release* do produto é criado, distribuído aos usuários e instalado em seu local de trabalho, no fluxo (*workflow*) de

A Análise e Projeto.
B Implantação.
C Gerenciamento de projeto.
D Implementação.
E Modelagem de negócios.

Exercícios

10. Assinale a alternativa que completa corretamente as três perspectivas do RUP:

A 1. Uma perspectiva topdown, que mostra as fases do modelo ao longo do tempo. 2. Uma perspectiva botonup, que mostra as atividades realizadas no processo. 3. Uma perspectiva prática, que sugere boas práticas a serem usadas durante o processo.

B 1. Uma perspectiva estática, que mostra as fases do modelo ao longo do tempo. 2. Uma perspectiva modular, que mostra as atividades realizadas no processo. 3. Uma perspectiva prática, que sugere boas práticas a serem usadas durante o processo.

C 1. Uma perspectiva dinâmica, que mostra as fases do modelo ao longo do tempo. 2. Uma perspectiva estática, que mostra as atividades realizadas no processo. 3. Uma perspectiva inserta, que sugere boas práticas a serem usadas durante o processo.

D 1. Uma perspectiva dinâmica, que mostra as fases do modelo ao longo do tempo.2. Uma perspectiva estática, que mostra as atividades realizadas no processo.3. Uma perspectiva prática, que sugere boas práticas a serem usadas durante o processo.