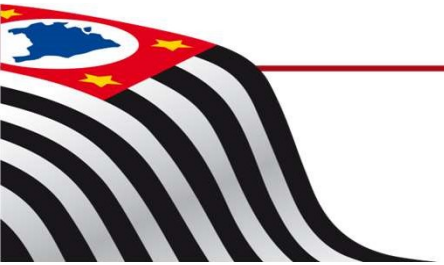
Engenharia de Software – Processos de Software

- Prof. Antonio Guardado

Sumário

- 1 – Motivações para uma Engenharia de Software
- 2 – Modelos de processos
- 3 – Modelo Cascata
- 4 – Modelo Evolucionário
- 5 – Espiral
- 6 – Técnicas de 4ª Geração
- 7 – Mudança na natureza do desenvolvimento de software
- 8 – Visão Genérica da ESW



1- Motivações para uma Engenharia de Software

- Software incorporado em praticamente todos os aspectos da nossa vida -> crescimento do número de pessoas interessadas nos recursos e funções oferecidas pelas aplicações. Portanto visões diferentes sobre funções e recursos

- *devemos realizar um esforço concentrado para compreender o problema antes de desenvolver uma solução de software*

1- Motivações para uma Engenharia de Software

- Requisitos da tecnologia da informação cada vez mais complexos. Mudança : ao invés de software desenvolvido por um único indivíduo temos grandes equipes.
- Software incorporado em diferentes ambientes (produtos eletrônicos, equipamentos médicos, sistemas de defesa e vigilância, etc.) -> Maior atenção para as interações de todos os elementos do sistema

- *Projetar (planejar) tornou-se uma atividade fundamental*

1- Motivações para uma Engenharia de Software

- Indivíduos, negócios e governos dependem, de forma crescente, de software para controlar atividades cotidianas e posteriormente tomar decisões táticas e estratégicas.
- Mal funcionamento do software -> pequenos inconvenientes até falhas catastróficas

- *Software deve apresentar qualidade elevada*

1- Motivações para uma Engenharia de Software

- Quanto melhor uma aplicação ⟹ maior base de usuários e maior longevidade
- Quanto mais usuários e longevidade ⟹ maior demanda por adaptação e aperfeiçoamento

- *Software deve ser passível de manutenção*

1- Motivações para uma Engenharia de Software

- Conclusão : Software, em todas as suas formas e em todos os campos de aplicação, deve passar pelos processos de engenharia : engenharia de software
- Relembrando : é o estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja confiável e que funcione eficientemente em máquinas reais.
- IEEE : (1) A aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, na operação e na manutenção de software; isto é, a aplicação de engenharia ao software. (2) O estudo de abordagens como definido em (1).

1.1 – Processos, métodos e ferramentas

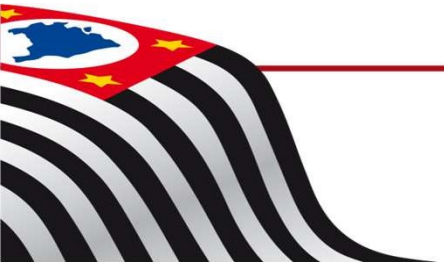
- Base são os processos (procedimentos)

1.1 – Processos, métodos e ferramentas

- Relembrando ...
- Método : fornecem as informações técnicas para desenvolver o software. COMO FAZER.
- Envolvem várias tarefas como Comunicação, Análise de requisitos, Modelagem de projeto, construção de programa, teste e suporte.
- Ferramenta : suporte automatizado para o processo e os métodos. Integração das ferramentas ⇨ Ferramentas CASE

1.1 – Processos, métodos e ferramentas

- Conjunto coerente de atividades, ações e tarefas na criação de algum produto de trabalho (work product).
- Atividade : esforço (trabalho) para atingir um objetivo amplo, por exemplo, comunicar-se com os interessados.
  - Independente do campo da aplicação, tamanho, complexidade, grau de rigor
- Ação : conjunto de tarefas que resultam num artefato de software (por exemplo, o projeto de arquitetura)
- Tarefa : objetivo pequeno, tangível e bem definido (por exemplo, realizar teste de unidade)



1.1 – Processos, métodos e ferramentas

- Metodologia de processo (*framework*) : estabelece um processo completo de engenharia por meio de atividades estruturais aplicáveis a todos os projetos.
- Além das atividades estruturais existem atividades de apoio (*umbrela activities*) aplicáveis em todo o processo.

1.1 – Processos, métodos e ferramentas

- Metodologia genérica compreende 5 atividades estruturais :
- 1 – Comunicação : comunicar-se e colaborar com os interessados. Compreender os objetivos das partes interessadas e realizar o levantamento das necessidades.
- 2 – Planejamento : Mapa que ajuda a guiar a equipe até o objetivo final. Fazer um plano de projeto que descreve as tarefas técnicas, riscos prováveis, recursos necessários, produtos resultantes e cronograma.

1.1 – Processos, métodos e ferramentas

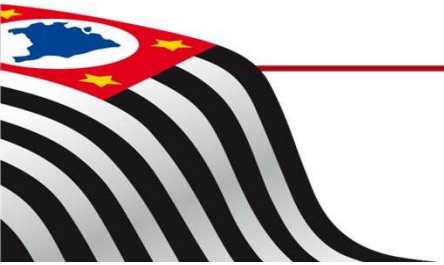
- 3 – Modelagem : Criar um esboço (modelo) de tal forma a entender o todo, como será a sua arquitetura, como os componentes interagem entre si e demais características. Refina-se o esboço detalhando cada vez mais.
- 4 – Construção : combina geração de código e testes para revelar erros e fazer ajustes.
- 5 – Emprego : O software (completo ou com incrementos parcialmente efetivados) é entregue aos interessados, que avaliam e fornecem feedback.

1.1 – Processos, métodos e ferramentas

- Atividades de apoio :
- 1 – Controle e acompanhamento do projeto : avaliação do progresso do plano de projeto tomando medidas para cumprir o cronograma.
- 2- Administração de riscos : avaliação de riscos que podem afetar o resultado ou a qualidade do produto/projeto.
- 3 – Garantia da qualidade de software : definção e condução das atividades que garantem a qualidade.

1.1 – Processos, métodos e ferramentas

- Atividades de apoio :
- 4 – Revisões Técnicas : avaliação dos artefatos da ESW, identificando e eliminando erros antes de se propagarem para a atividade seguinte.
- 5 – Medição : definição e coleta das medidas (de produto, projeto e de processo). Auxilia na entrega do software conforme os requisitos.
- 6 – Gerenciamento da configuração de software : gerência dos efeitos das mudanças ao longo do processo.



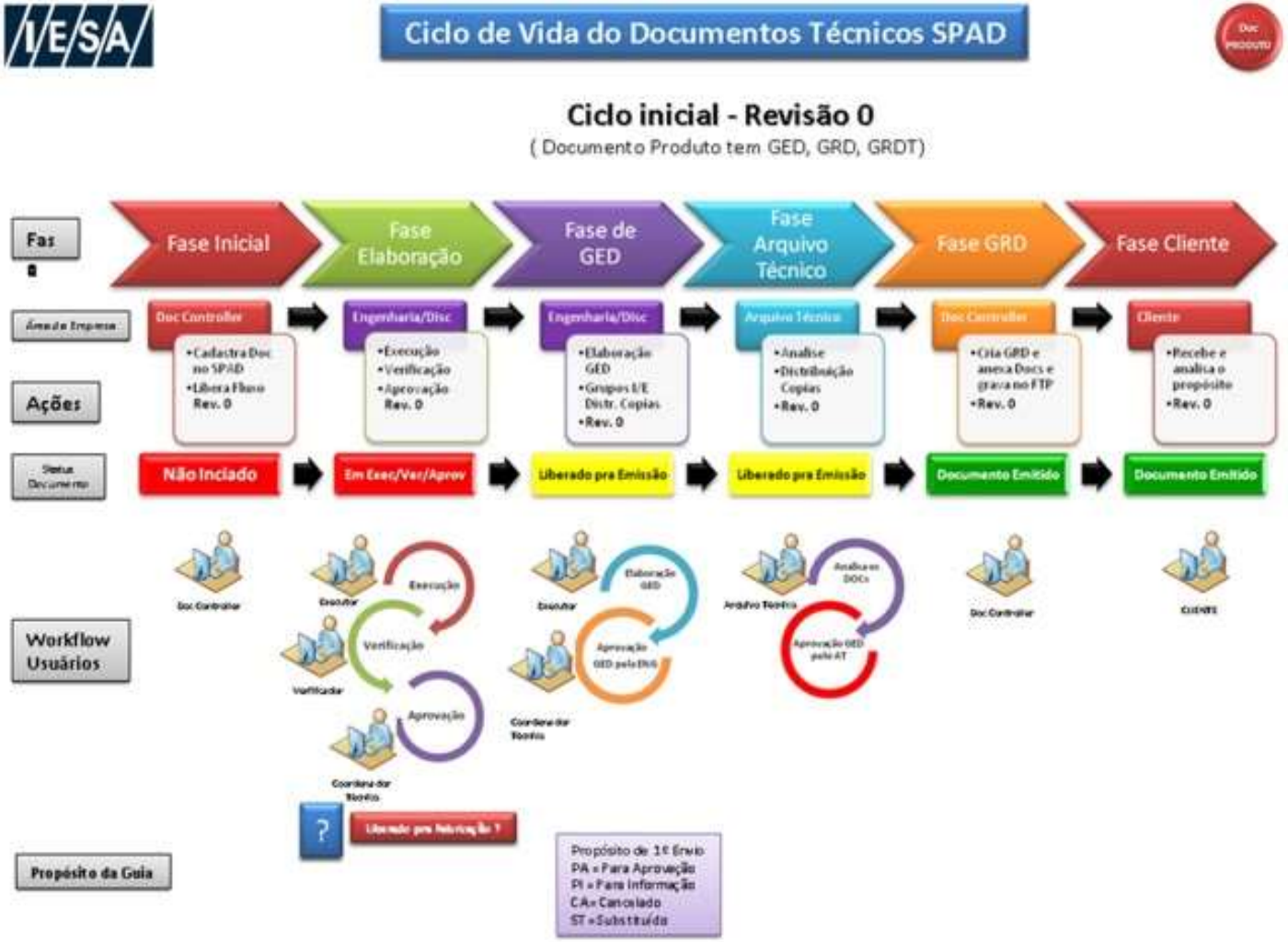
1.1 – Processos, métodos e ferramentas

- Atividades de apoio :
- 7 – Gerenciamento da reusabilidade : definição de critérios para o reuso de artefatos e de mecanismos para a obtenção de componentes reutilizáveis.
- 8- Preparo e produção de artefatos de software : engloba as atividades para criar artefatos : modelos, documentos, logs, formulários, etc.

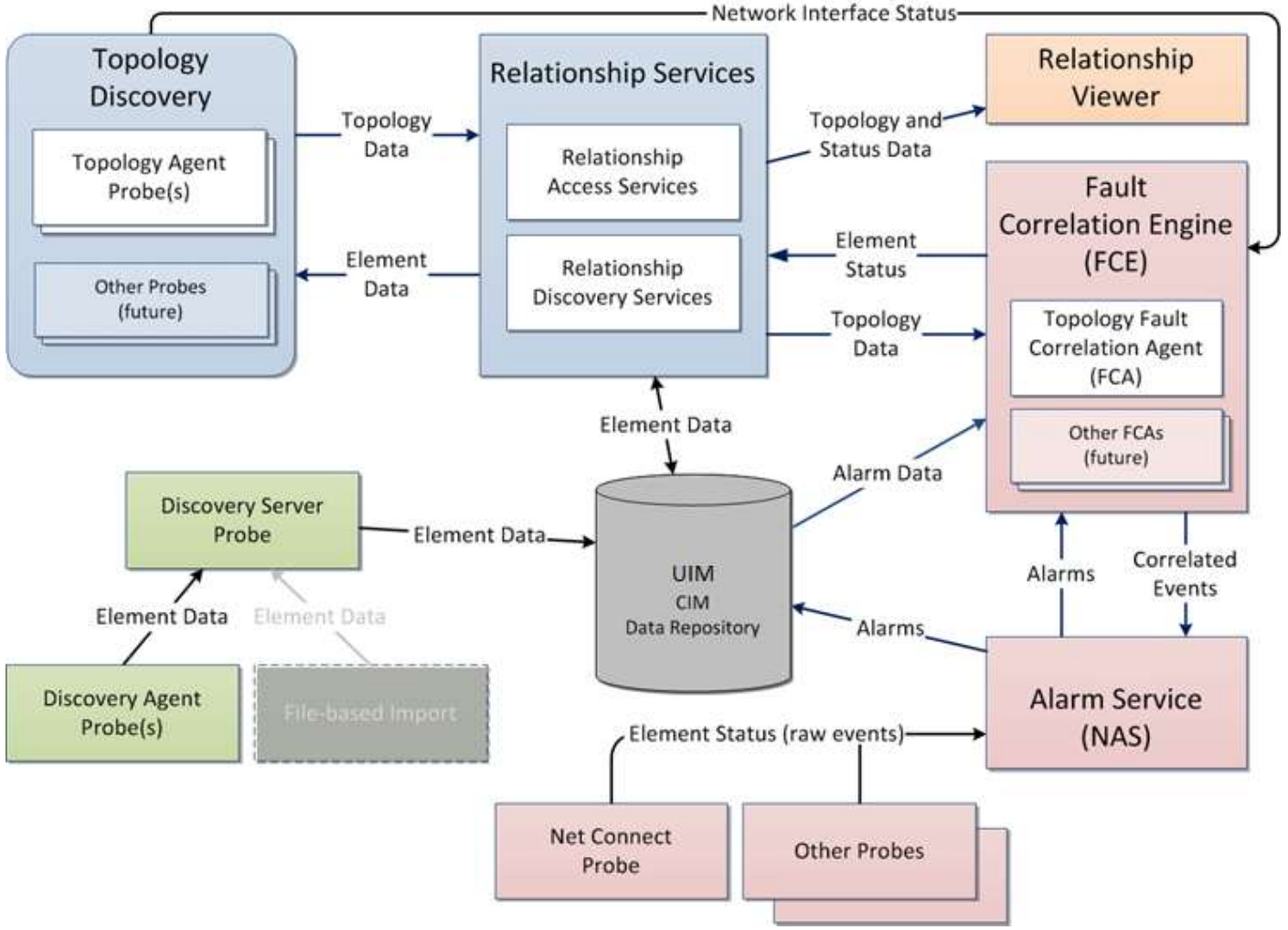
2- Modelo de Processo

- Descrição simplificada de um processo de software a partir de uma perspectiva específica
  - Modelo de Workflow : sequência de atividades no processo, entradas, saídas e dependências. Atividades representam ações humanas
  - Modelo de fluxo de dados ou de atividade : conjunto de atividades que realizam a transformação dos dados. Atividades são transformações realizadas por pessoas ou computadores
  - Modelo de papel/ação : papéis das pessoas envolvidas e as atividades pelas quais são responsáveis

Modelo de Workflow : sequência de atividades no processo, entradas, saídas e dependências. Atividades representam ações humanas



Modelo de fluxo de dados ou de atividade : conjunto de atividades que realizam a transformação dos dados. Atividades são transformações realizadas por pessoas ou computadores

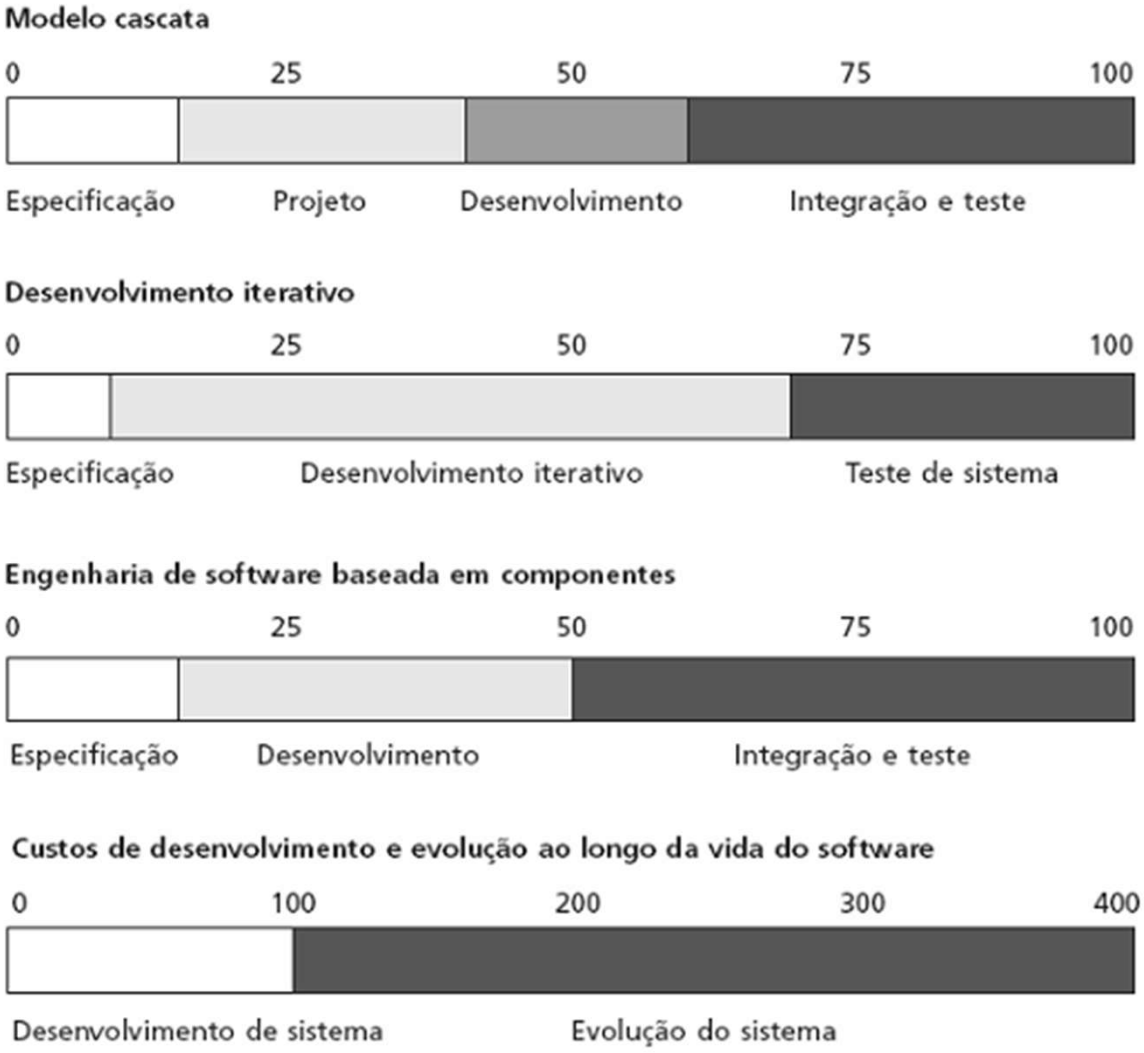


Modelo de papel/ação : papéis das pessoas envolvidas e as atividades pelas quais são responsáveis

2- Quais são os custos da engenharia de software?

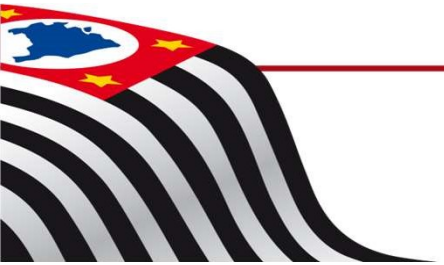
- 60% dos custos são custos de desenvolvimento
- 40% são custos de testes
- Para software sob encomenda, os custos de evolução normalmente excedem os de desenvolvimento
- Os custos variam dependendo do tipo de sistema que está sendo desenvolvido e dos requisitos do sistema, tais como desempenho e confiabilidade
- A distribuição de custos depende do modelo de desenvolvimento que é usado

2- Distribuição de custos nas atividades

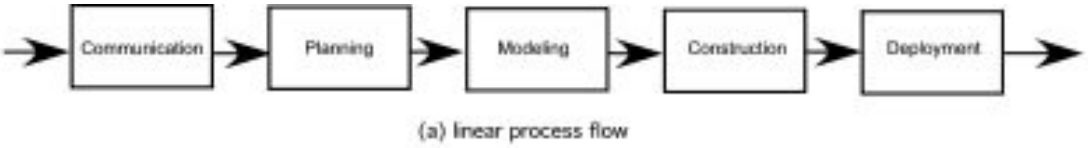


2.1 – Critérios para escolha de um Modelo de Processo de software

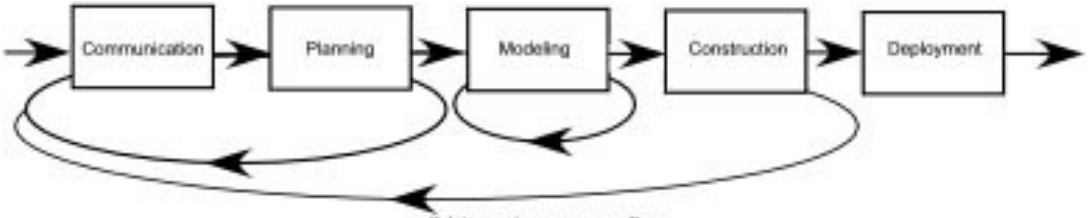
- natureza do projeto e da aplicação
- métodos e ferramentas a serem usados
- controles e produtos que precisam ser entregues



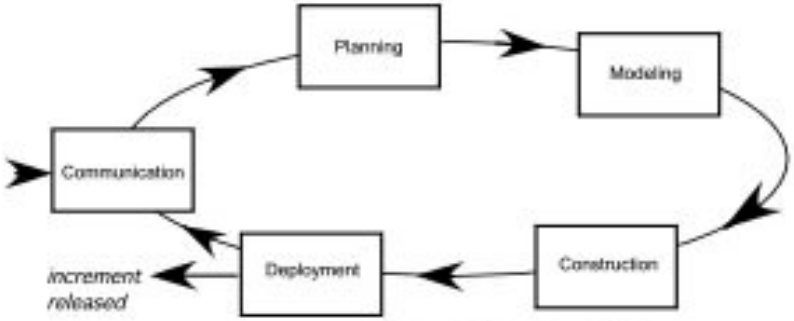
2.2 – Fluxo de Processo



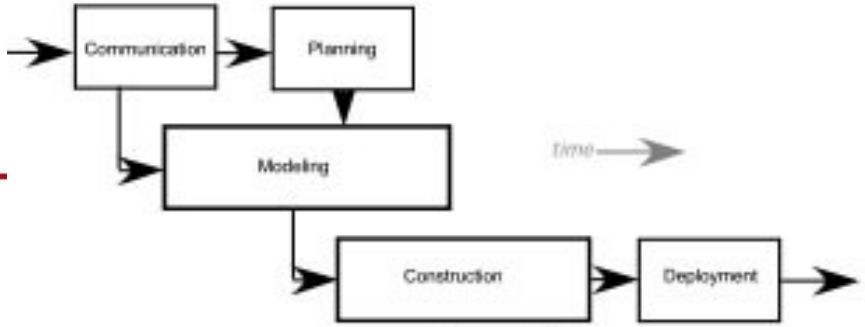
(a) linear process flow



(b) iterative process flow



(c) evolutionary process flow



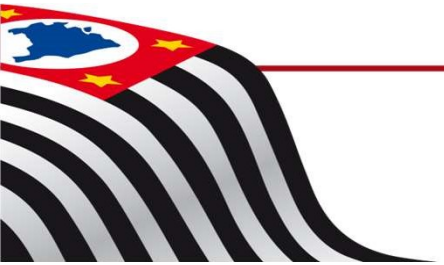
(d) parallel process flow

3 – Modelos de Processos Prescritivos

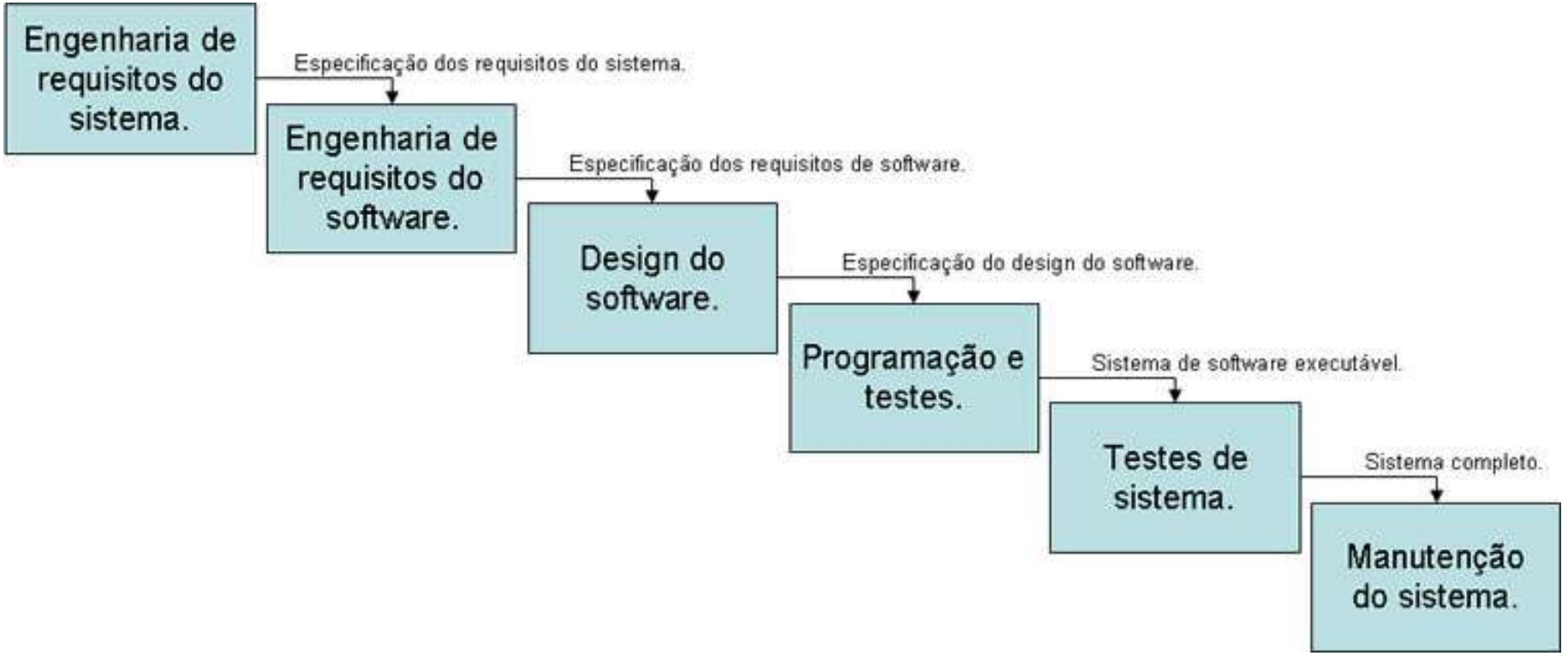
Um **Modelo Prescritivo** de Processo de **Software** é um conjunto de elementos que inclui ações de **engenharia de software**, produtos de trabalho (artefatos) e mecanismos que garantam a qualidade e controle de modificações em cada projeto necessárias para o desenvolvimento de um sistema de **software** (PRESSMAN, 2010). Prescrevem um conjunto de elementos e a maneira como esses elementos se inter-relacionam.

3.1- Ciclo de Vida Clássico (Cascata)

- Linear e sequencial
- modelo mais antigo e o mais amplamente usado da engenharia de software
- modelado em função do ciclo da engenharia convencional
- requer uma abordagem sistemática, seqüencial ao desenvolvimento de software



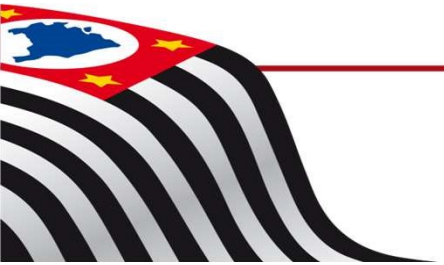
3.1- Cascata (*waterfall*)



3.1.1 - Atividades do Ciclo de Vida Clássico

1- ANÁLISE E ENGENHARIA DE SISTEMAS

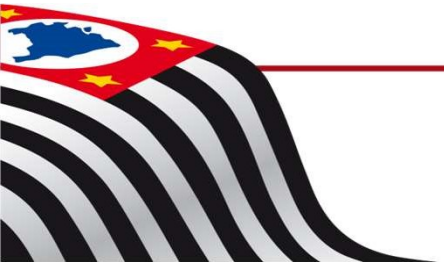
- envolve a coleta de requisitos em nível do sistema, com uma pequena quantidade de projeto e análise de alto nível
- esta visão é essencial quando o software deve fazer interface com outros elementos (hardware, pessoas e banco de dados)



3.1.1 - Atividades do Ciclo de Vida Clássico

2- ANÁLISE DE REQUISITOS DE SOFTWARE

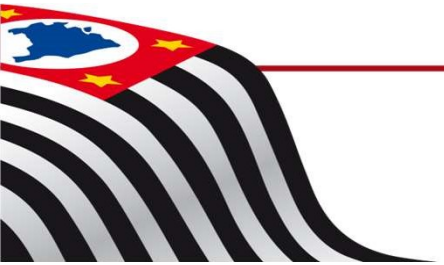
- o processo de coleta dos requisitos é intensificado e concentrado especificamente no software
- deve-se compreender o domínio da informação, a função, desempenho e interfaces exigidos
- os requisitos (para o sistema e para o software) são documentados e revistos com o cliente



3.1.1 - Atividades do Ciclo de Vida Clássico

3- PROJETO (Design)

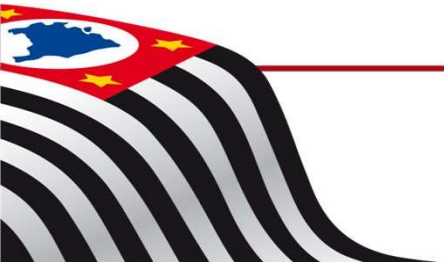
- tradução dos requisitos do software para um conjunto de representações que podem ser avaliadas quanto à qualidade, antes que a codificação se inicie
- se concentra em 4 atributos do programa:
  - *Estrutura de Dados,*
  - *Arquitetura de Software*,
  - *Detalhes Procedimentais* e
  - *Caracterização de Interfaces*



3.1.1 - Atividades do Ciclo de Vida Clássico

4- CODIFICAÇÃO

- tradução das representações do projeto para uma linguagem “artificial” resultando em instruções executáveis pelo computador

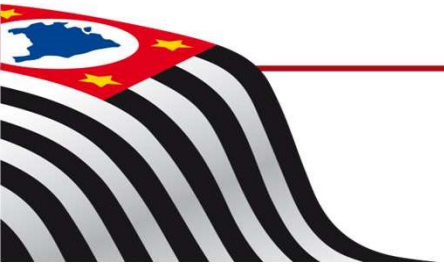


3.1.1 - Atividades do Ciclo de Vida Clássico

5- TESTES

Concentra-se:

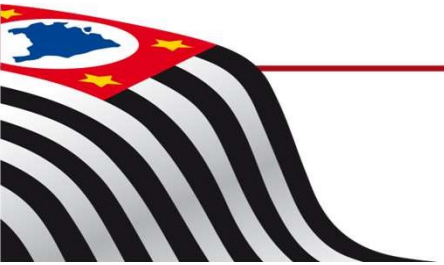
- nos aspectos lógicos internos do software, garantindo que todas as instruções tenham sido testadas
- nos aspectos funcionais externos, para descobrir erros e garantir que a entrada definida produza resultados que concordem com os esperados.



3.1.1 - Atividades do Ciclo de Vida Clássico

6- MANUTENÇÃO

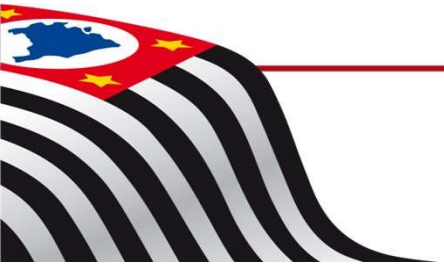
- provavelmente o software deverá sofrer mudanças depois que for entregue ao cliente
- causas das mudanças: e*rros*, *adaptação do software para acomodar mudanças em seu ambiente externo* e *exigência do cliente para acréscimos funcionais e de desempenho*



3.1.2 - Problemas com o Ciclo de Vida Clássico

- O mais antigo e amplamente usado.
- Projetos reais raramente seguem o fluxo seqüencial que ele propõe. Ocorrem iterações que trazem problemas na aplicação do paradigma.
- É difícil para o cliente declarar todas as exigências explicitamente. É difícil acomodar as incertezas naturais que existem no começo de muitos projetos.
- O cliente deve ter paciência. Uma versão do sw só estará disponível em um ponto tardio do cronograma. Um erro crasso, pode ser desastroso.
- Só é apropriado quando os requisitos são bem conhecidos.

- ✓ Embora o Ciclo de Vida Clássico tenha fragilidades, ele é significativamente melhor do que uma abordagem casual ao desenvolvimento de software



4 – Desenvolvimento evolucionário

- Desenvolvimento exploratório : trabalhar com o cliente para explorar seus requisitos e entregar um sistema final. Inicia com partes compreendidas e incrementa-se com outras partes
- Protótipos descartáveis : compreender os requisitos do cliente e desenvolver uma melhor definição de requisitos para o sistema. Protótipos para fazer experimentos com parte dos requisitos que estejam mal entendidas

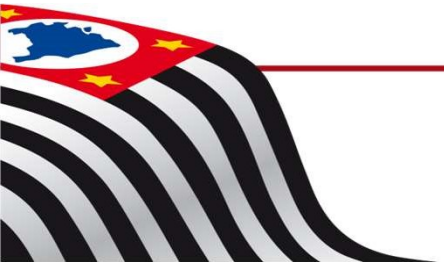
4 – Desenvolvimento evolucionário

Problemas

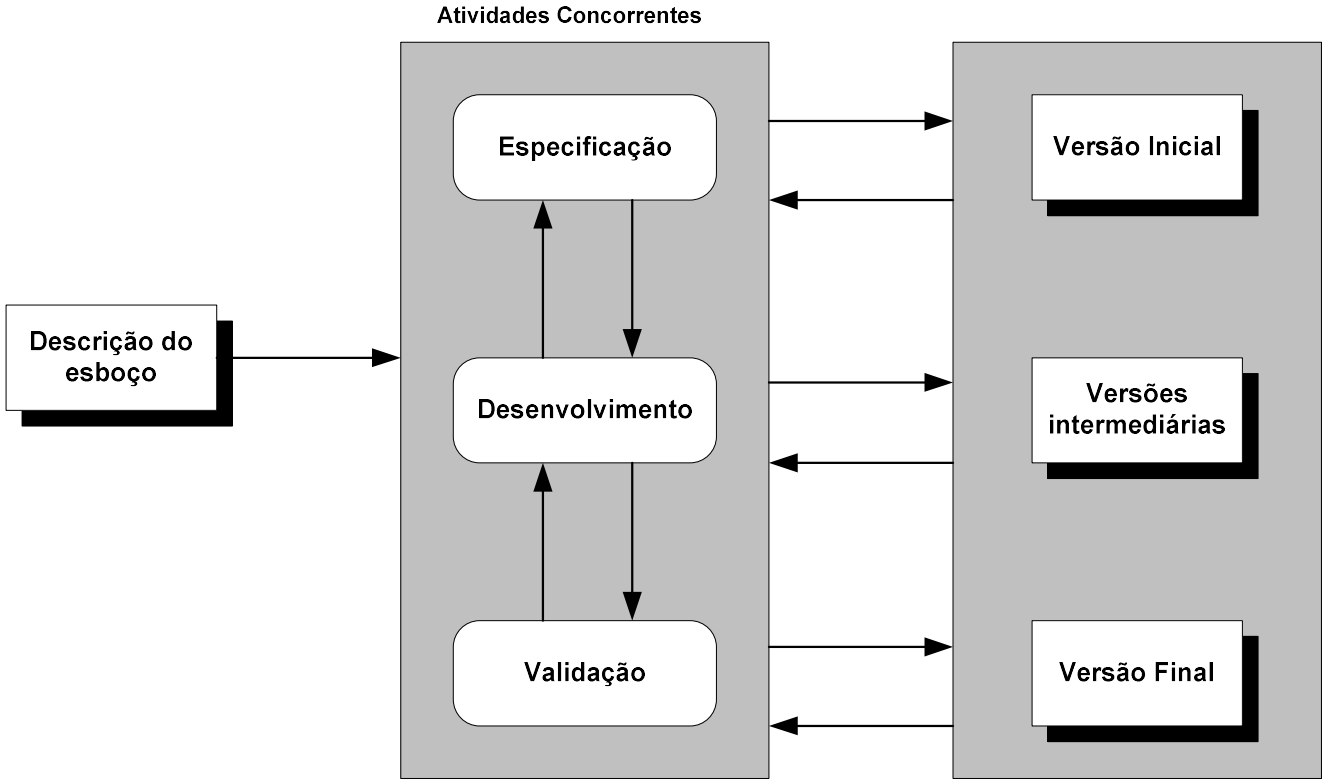
- Falta de visibilidade sobre o processo.
- Sistema geralmente pouco estruturado.
- Habilidades especiais (ex. em linguagens para uma rápida prototipação) são requeridas.

Aplicabilidade

- Para sistemas interativos pequenos ou médios.
- Para partes de de grandes sistemas (ex. A interface com o usuário).
- Para sistemas com pouco tempo de vida.

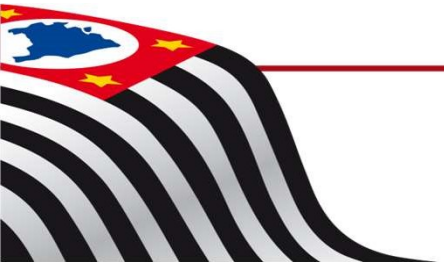


4 - Desenvolvimento Evolucionário



4.1 - Prototipação

- processo que 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 definiu um conjunto de objetivos gerais para o software, mas não identificou requisitos de entrada, processamento e saída com detalhes.*

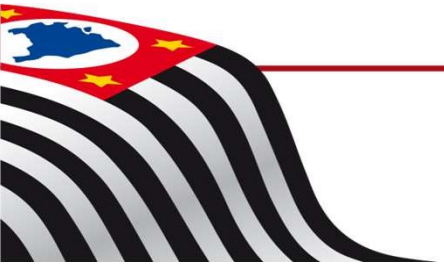


4.1- Prototipação

4.1.1 - Atividades da Prototipação

1- *OBTENÇÃO DOS REQUISITOS:* desenvolvedor e cliente definem os objetivos gerais do software, identificam quais requisitos são conhecidos e as áreas que necessitam de definições adicionais.

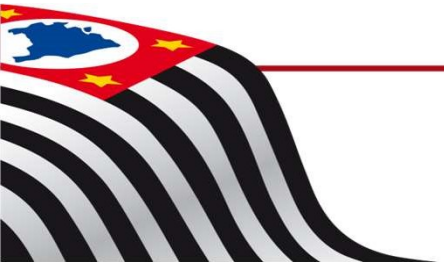
2- *PROJETO RÁPIDO:* representação dos aspectos do software que são visíveis ao usuário (abordagens de entrada e formatos de saída)



4.1.1 - Atividades da Prototipação

3- *CONSTRUÇÃO PROTÓTIPO:* implementação do projeto rápido

4- *AVALIAÇÃO DO PROTÓTIPO:* cliente e desenvolvedor avaliam o protótipo



4.1.1 - Atividades da Prototipação

5- REFINAMENTO DOS REQUISITOS: cliente e desenvolvedor refinam os requisitos do software a ser desenvolvido. Ocorre neste ponto um *processo de iteração* que pode conduzir a atividade 1 até que as necessidades do cliente sejam satisfeitas e o desenvolvedor compreenda o que precisa ser feito.

6- CONSTRUÇÃO PRODUTO: identificados os requisitos, o protótipo deve ser descartado e a versão de produção deve ser construída considerando os critérios de qualidade.

4.1.2 - Problemas com a Prototipação

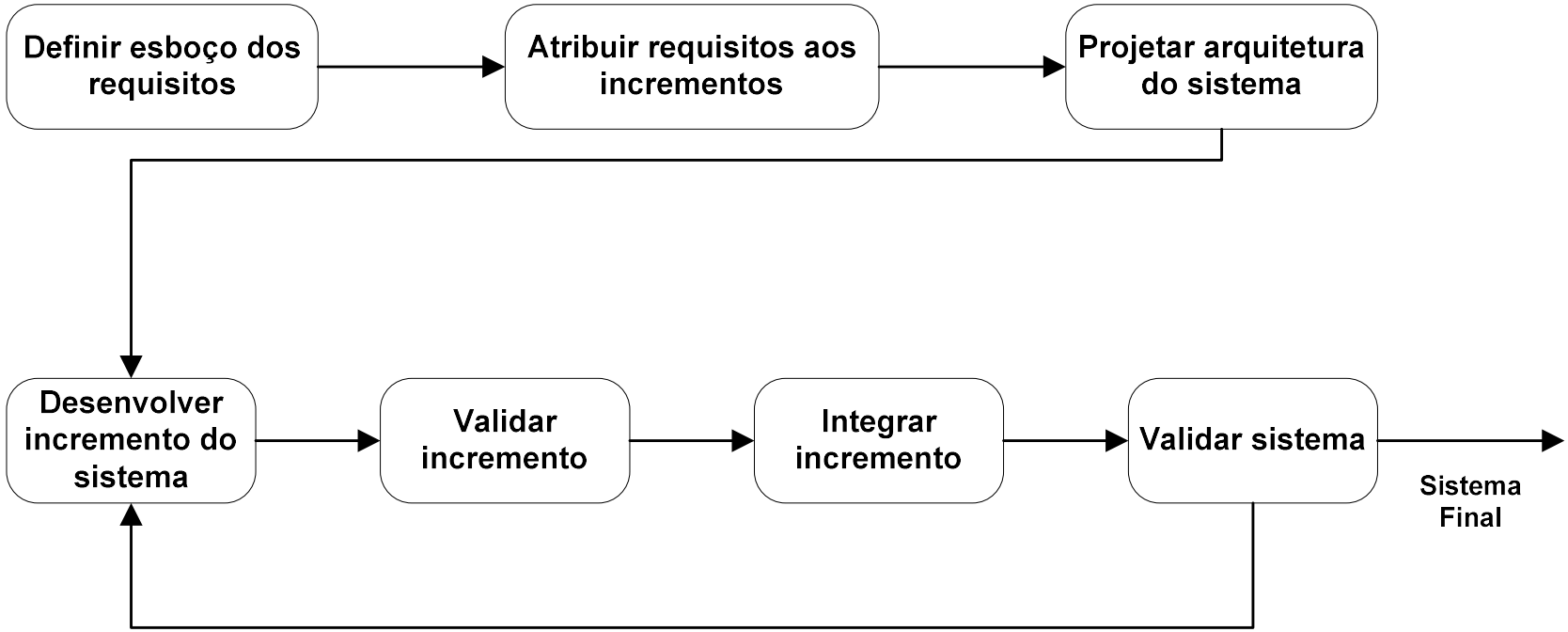
- cliente não sabe que o software que ele vê não considerou, durante o desenvolvimento, a qualidade global e a manutenibilidade a longo prazo. Não aceita bem a idéia que a versão final do software vai ser construída e "força" a utilização do protótipo como produto final
- desenvolvedor freqüentemente faz uma implementação comprometida (utilizando o que está disponível) com o objetivo de produzir rapidamente um protótipo. Depois de um tempo ele se familiariza com essas escolhas, e esquece que elas não são apropriadas para o produto final.

- *ainda que possam ocorrer problemas, a prototipação é um ciclo de vida eficiente.*
- *a chave é definir-se 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 a fim de definir os requisitos.*

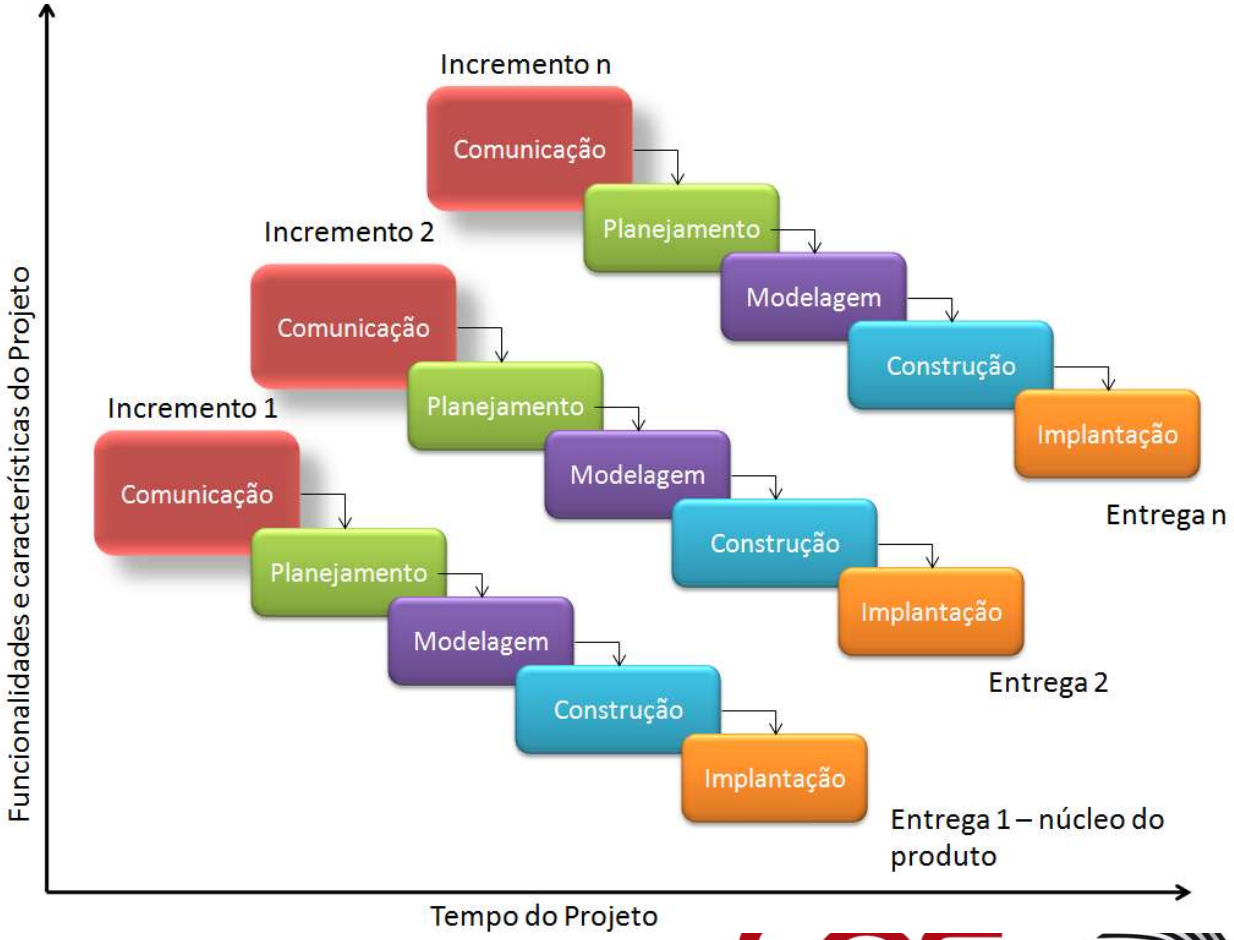
4.2 – Iteração de Processo

- Os requisitos do sistema sempre evoluem ao longo do projeto, então o processo de iteração dos estágios anteriores é retrabalhado e vira parte do processo para grandes sistemas.
- Iteração pode ser aplicada a qualquer modelo genérico de ciclo de vida.
  - 1 – desenvolvimento incremental : especificação, projeto e implementação são desdobradas em uma série de estágios, que por sua vez são desenvolvidos;
  - 2- Espiral : evolui de um esboço inicial para o sistema final : Análise de riscos

4.3 – Desenvolvimento incremental



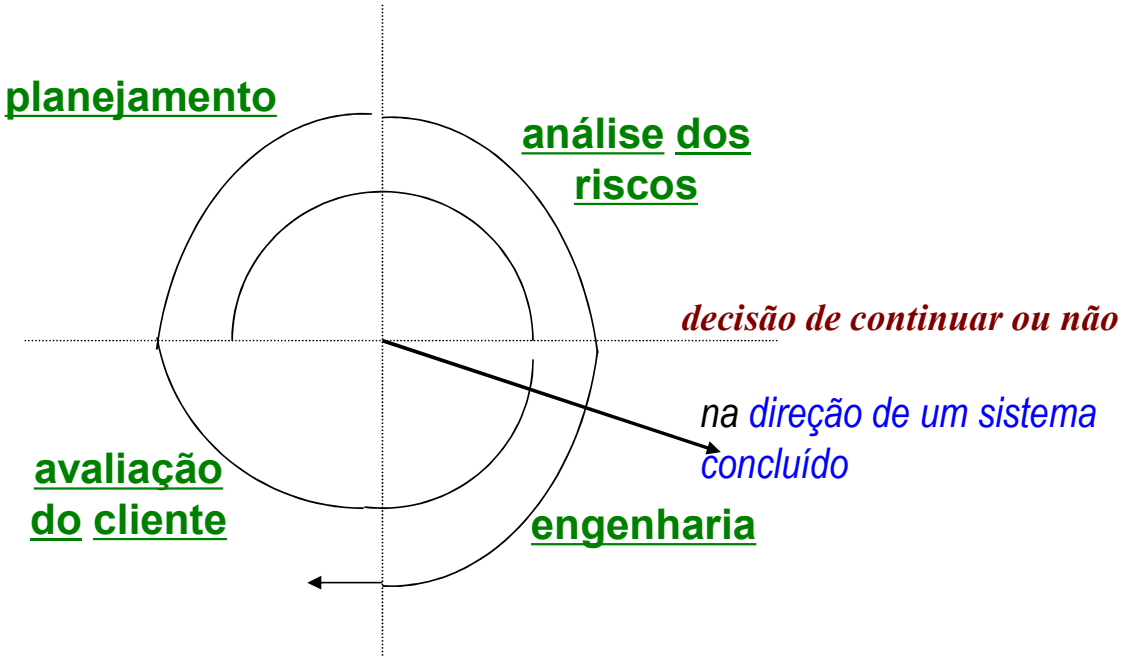
4.3 – Desenvolvimento incremental



5 – Modelo Espiral

- O Processo é representado como um espiral ao invés de uma sequência de atividades com voltas para trás.
- Cada volta no espiral representa uma fase no processo.
- Não há fases fixas como especificação ou projeto. As voltas na espiral são escolhidas dependendo do que é requisitado
- Os riscos são explicitamente avaliados e resolvidos durante todo o processo

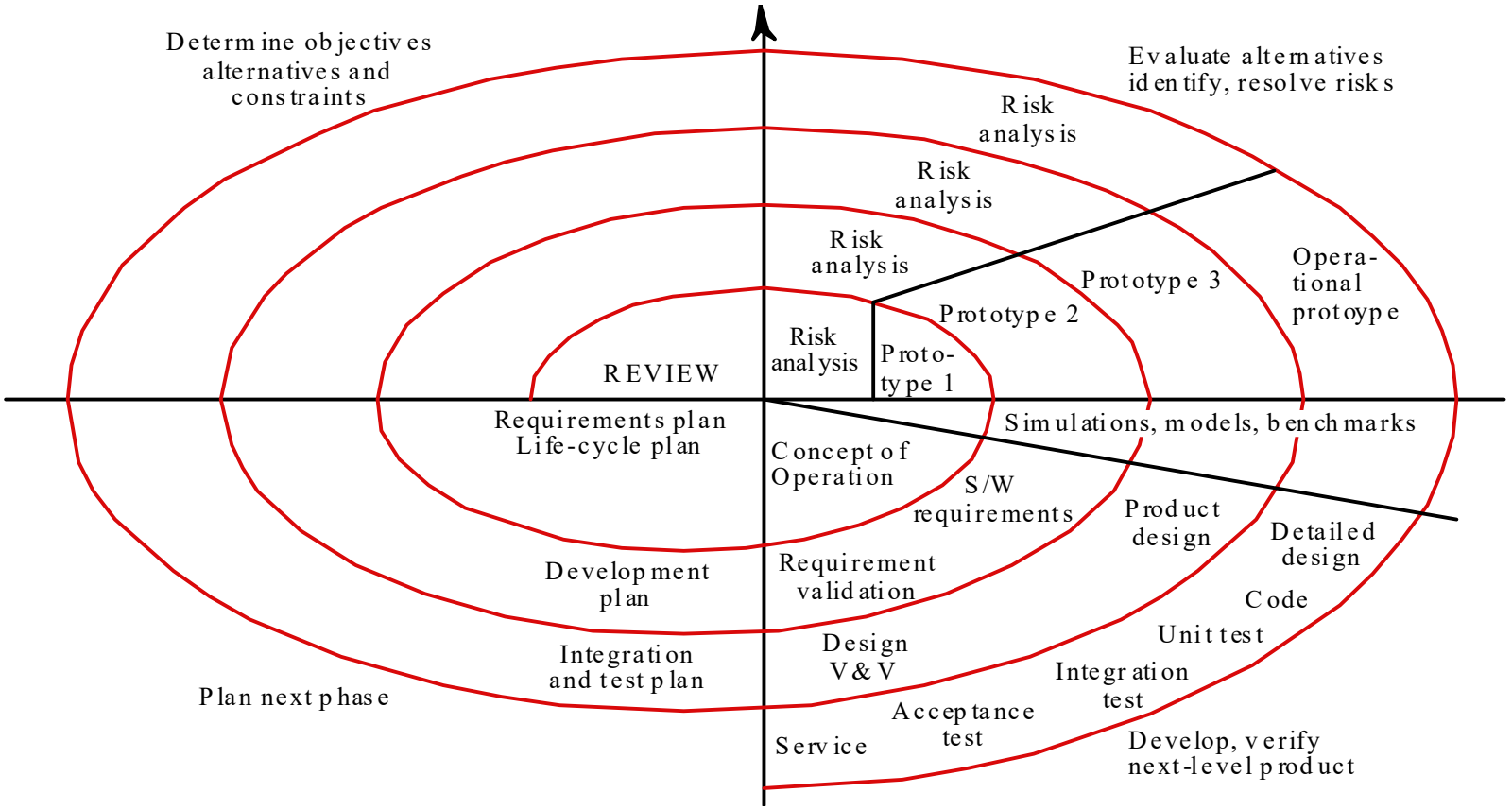
5 – Modelo Espiral



5 - Ciclo de Vida em Espiral (Bohem 1988)

- engloba as melhores características do ciclo de vida Clássico e da Prototipação, adicionando um novo elemento: a *Análise de Risco*
- segue a abordagem de passos sistemáticos do *Ciclo de Vida Clássico* 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

5 – Modelo Espiral - Bohem



5.1- Atividades do Ciclo de Vida em Espiral

1- *PLANEJAMENTO:* determinação dos objetivos, alternativas e restrições

2- *ANÁLISE DE RISCO:* análise das alternativas e identificação / resolução dos riscos

3- *CONSTRUÇÃO (ENGENHARIA) :* desenvolvimento do produto no nível seguinte

4- *AVALIAÇÃO DO CLIENTE:* avaliação do produto e planejamento das novas fases

5.2 - Comentários sobre o Ciclo de Vida em Espiral

- é, atualmente, a abordagem mais realística 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

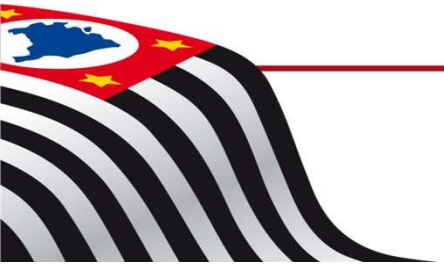
6- Técnicas de 4ª Geração

Concentra-se na capacidade de se especificar o software a uma máquina em um nível que esteja próximo à linguagem natural.

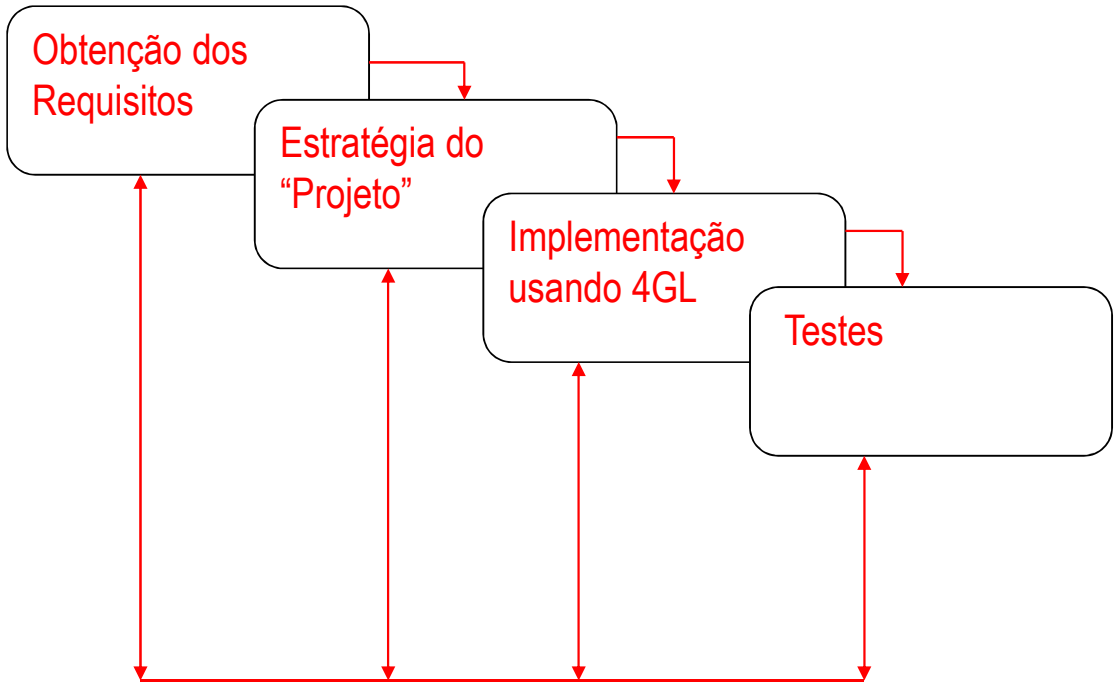
Engloba um conjunto de ferramentas de software que possibilitam que:

⇨ *o sistema seja especificado em uma linguagem de alto nível e*

⇨*o código fonte seja gerado automaticamente a partir dessas especificações*



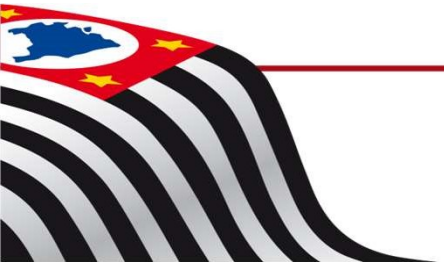
6 – Técnicas de 4ª Geração



6.1 - Ferramentas do ambiente de desenvolvimento de software de 4ª Geração

O ambiente de desenvolvimento de software que sustenta o ciclo de vida de 4ª geração inclui as 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



6.2 - Atividades das Técnicas de 4ª Geração

1- 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 um modo que uma ferramenta 4GL possa consumir
- as 4GLs atuais não são sofisticadas suficientemente para acomodar a verdadeira "linguagem natural"

6.2 - Atividades das Técnicas de 4ª Geração

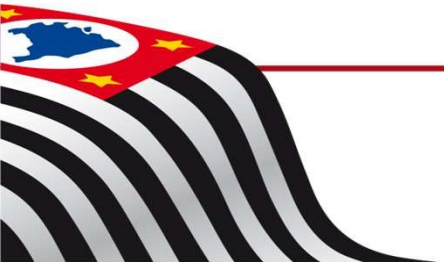
2- ESTRATÉGIA DE "PROJETO": para pequenas aplicações é possível mover-se do passo de Obtenção dos Requisitos para o passo de Implementação usando uma *linguagem de quarta geração*

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

6.2 - Atividades das Técnicas de 4ª Geração

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

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

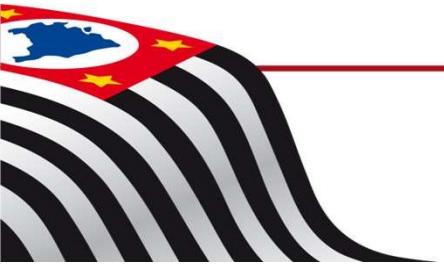


6.3 - Comentários sobre as Técnicas de 4ª Geração

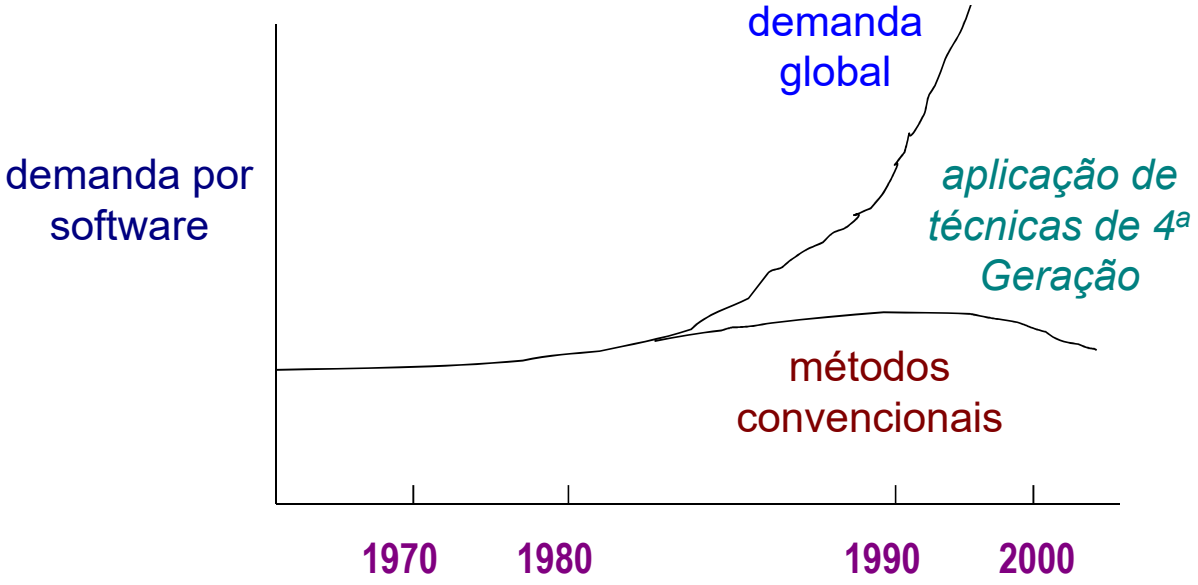
PROPONENTES: redução dramática no tempo de desenvolvimento do software (aumento de produtividade)

OPONENTES: as 4GL atuais não são mais fáceis de usar do que as linguagens de programação

- ⇨ *o código fonte produzido é ineficiente*
- ⇨ *a manutenibilidade de sistemas usando técnicas 4G ainda é questionável*

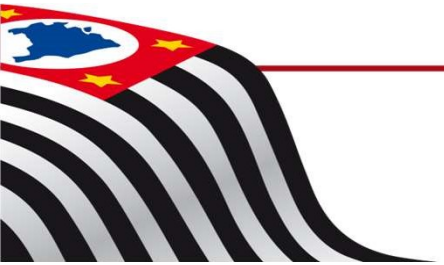


7 - Mudança na natureza de desenvolvimento de software

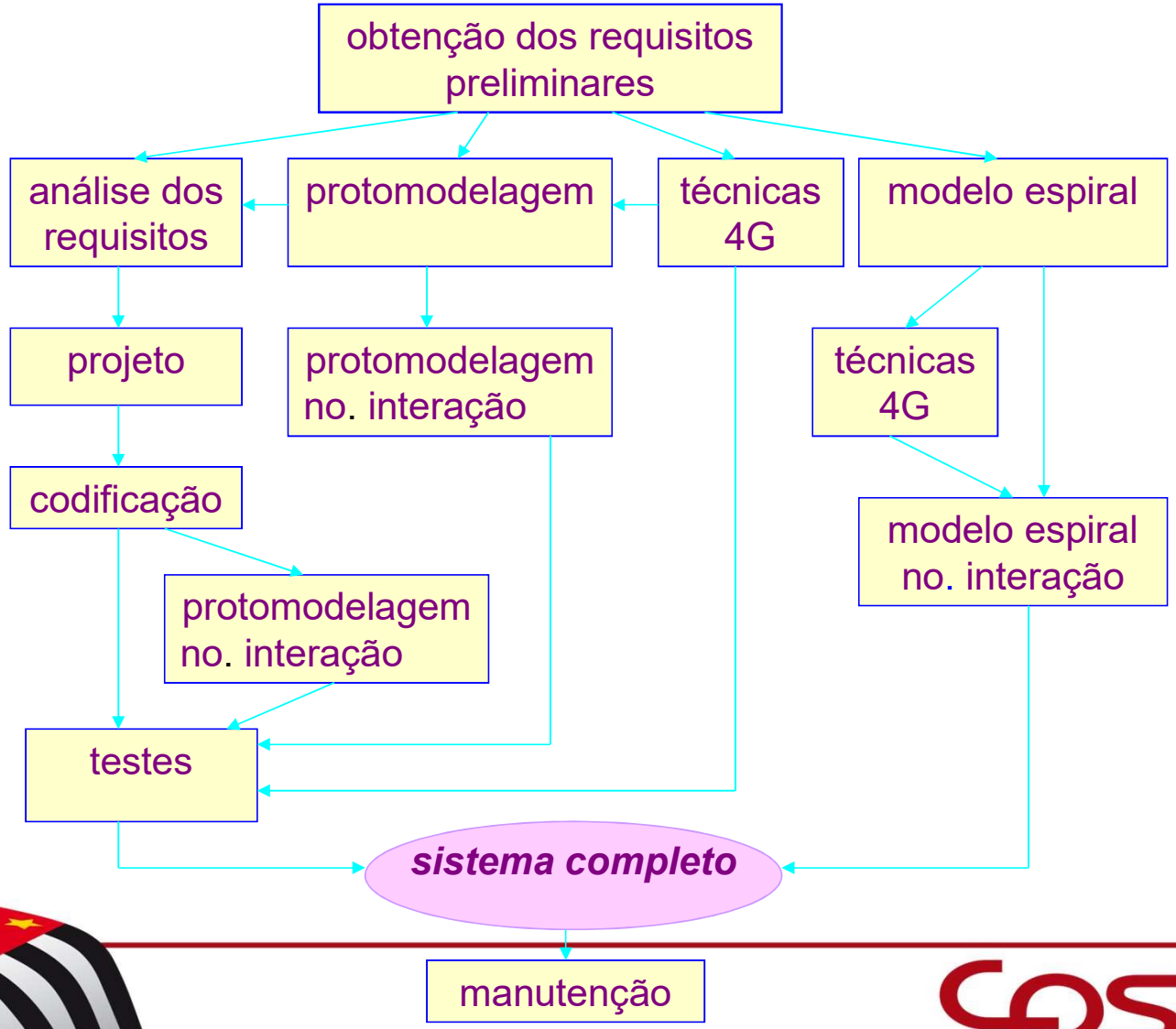


7.1 – Outros modelos de Processo

- RAD (Rapid Application Development)
- Processo formal
- Modelo de componentes
- Processo Unificado
- Métodos Ágeis
- Orientado a Aspectos



Combinação dos Métodos de Ciclo de Vida



8 - Engenharia de Software *uma visão genérica*

O processo de desenvolvimento de software contém 3 fases genéricas, independentes do modelo de engenharia de software escolhido:

- ☆ DEFINIÇÃO,
- ☆ DESENVOLVIMENTO e
- ☆ MANUTENÇÃO.

8 - Engenharia de Software *uma visão genérica*

FASE DE DEFINIÇÃO: *"o que"* será desenvolvido.

- *Análise do Sistema*: define o papel de cada elemento num sistema baseado em computador, atribuindo em última análise, o papel que o software desempenhará.
- *Planejamento do Projeto de Software*: assim que o escopo do software é estabelecido, os riscos são analisados, os recursos são alocados, os custos são estimados e, tarefas e programação de trabalho definidas.
- *Análise de Requisitos*: o escopo definido para o software proporciona uma direção, mas uma definição detalhada do domínio da informação e da função do software é necessária antes que o trabalho inicie.

8 - Engenharia de Software *uma visão genérica*

DESENVOLVIMENTO: "como" o software vai ser desenvolvido.

- *Projeto de Software*: traduz os requisitos do software num conjunto de representações (algumas gráficas, outras tabulares ou baseadas em linguagem) que descrevem a estrutura de dados, a arquitetura do software, os procedimentos algoritmicos e as características de interface.
- *Codificação*: as representações do projeto devem ser convertidas numa linguagem artificial (a linguagem pode ser uma linguagem de programação convencional ou uma linguagem não procedimental) que resulte em instruções que possam ser executadas pelo computador.
- *Realização de Testes do Software*: logo que o software é implementado numa forma executável por máquina, ele deve ser testado para que se possa descobrir defeitos de função, lógica e implementação.

8 - Engenharia de Software *uma visão genérica*

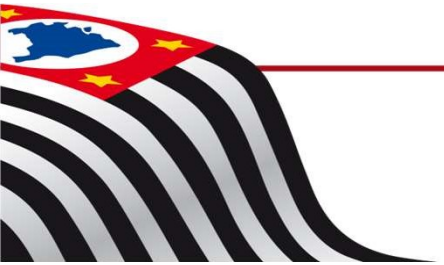
FASE DE MANUTENÇÃO: concentra-se nas "mudanças" que ocorrerão depois que o software for liberado para uso operacional

- *Correção*
- *Adaptação*
- *Melhoramento Funcional*

8 - Engenharia de Software *uma visão genérica*

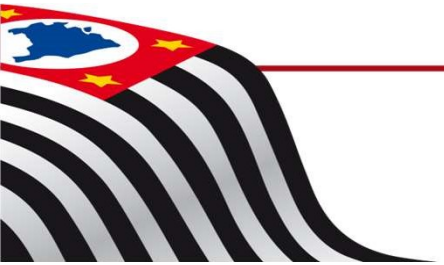
Correção: mesmo com as melhores atividades de garantia de qualidade de software, é provável que o cliente descubra defeitos no software. A manutenção *corretiva* muda o software para corrigir defeitos.

Adaptação: com o passar do tempo, o ambiente original (por exemplo a CPU, o sistema operacional e periféricos) para o qual o software foi desenvolvido provavelmente mudará. A manutenção *adaptativa* muda o software para acomodar mudanças em seu ambiente.



8 - Engenharia de Software *uma visão genérica*

Melhoramento Funcional: a medida que o software é usado, o cliente/usuário reconhecerá funções adicionais que oferecerão benefícios. A *manutenção perfectiva* estende o software para além de suas exigências funcionais originais.



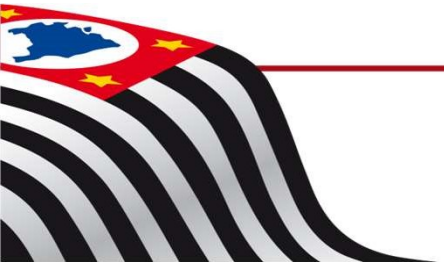
8 - Engenharia de Software *uma visão genérica*

ATIVIDADES DE PROTEÇÃO as fases e etapas correlatas descritas são complementadas por uma série de atividades de proteção.

Revisões: efetuadas para garantir que a qualidade seja mantida à medida que cada etapa é concluída.

Documentação: é desenvolvida e controlada para garantir que informações completas sobre o software estejam disponíveis para uso posterior.

Controle das Mudanças: é instituído de forma que as mudanças possam ser aprovadas e acompanhadas.



Conclusão

ENGENHARIA DE SOFTWARE

pode ser vista como uma abordagem de desenvolvimento de software elaborada com disciplina e métodos bem definidos.

....."a construção por múltiplas pessoas de um software de múltiplas versões" [Parnas 1987]