**MODELO DE DESENVOLVIMENTO DE SOFTWARE (CICLO DE VIDA)**

É uma representação abstrata do processo de desenvolvimento que define como as etapas relativas ao desenvolvimento de software serão conduzidas e interrelacionadas para atingir o objetivo do desenvolvimento que é a obtenção de um produto de software de alta qualidade a um custo relativamente baixo.

**para escolha de um Ciclo de Vida de Software:**

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

**Modelos de ciclo de vida de software**

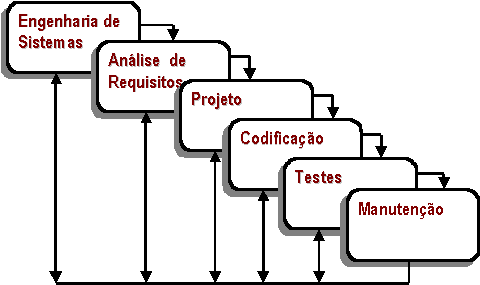
Podem-se citar como exemplos de modelo de

ciclo de vida de software:

* **Modelo em cascata,**
* **Modelo incremental,**
* **Modelo de prototipação,**
* **Modelo espiral**
* **Desenvolvimento formal**
* **Desenvolvimento rápido (RAD)**
* **Desenvolvimento com ferramentas CASE**

**Ciclo de Vida Clássico  (Cascata/ Queda dágua)**

* 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

****

**Atividades do ciclo de vida clássico**

1. **Analise e engenharia de sistemas** Como o software faz parte de um sistema mais amplo, esta fase envolve a coleta de requisitos em nível de sistema em alto nível. Trata-se da visão do sistema como um todo no qual o software participa (analise de visão e escopo)
2. **Análise e definição de requisitos (conceitual)**

As funções, restrições e os objetivos do sistema são estabelecidos por consultas aos futuros usuários. Em seguida são definidas em detalhes e servem como especificação do sistema.

1. **Projeto (especificações físicas)**

Envolve a 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 e**
        + **Caracterização de Interfaces**

1. **Codificação**

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

1. **Testes**

Concentram-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.

1. **Implantação** – colocar em funcionamento na estrutura organizacional (se for o caso)
2. **Manutençao**

O software deverá sofrer mudanças depois que for entregue ao cliente (evolução)

**Problemas com o Ciclo de Vida Clássico**

* projetos reais raramente seguem o fluxo seqüencial que o modelo propõe. **Em principio cada fase resulta numa série de documentos que são aprovados (assinados). A fase seguinte não deve ser iniciada antes que a fase anterior tenha sido concluída. Na prática estes estágios se sobrepõem e trocam informações entre si.**
* 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 numa etapa avançada do desenvolvimento

**Clássico (comentários)**

*Embora o Ciclo de Vida Clássico tenha fragilidades, ele é significativamente melhor do que uma abordagem casual ao desenvolvimento de software. É aconselhado quando os requisitos são bem compreendidos – contudo reflete a prática da engenharia. É bastante utilizado em grandes projetos de software.*

**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.

*É 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.*

***fim***

***início***

**construção produto**

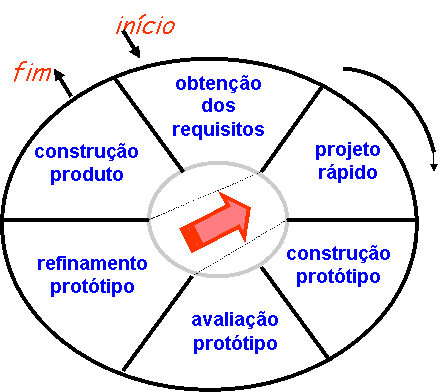
**refinamento protótipo**

**avaliação protótipo**

**construção protótipo**

**projeto rápido**

**obtenção dos requisitos**

****

**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

1. **Projeto Rápido:**  representação dos aspectos do software que são visíveis ao usuário (abordagens de entrada e formatos de saída)
2. **Construção Protótipo:** implementação  do projeto rápido

1. **Avaliação do Protótipo:** cliente e desenvolvedor avaliam o protótipo
2. **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 1a. atividade até que as necessidades do cliente sejam satisfeitas e o desenvolvedor compreenda o que precisa ser feito.

**Problemas com a Prototipação**

O 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.

**Comentário**

*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.*

**Ciclo de Vida em Espiral**

* 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 aPrototipação***,*** em qualquer etapa da evolução do produto, como mecanismo de redução de riscos

***decisão de continuar ou não***

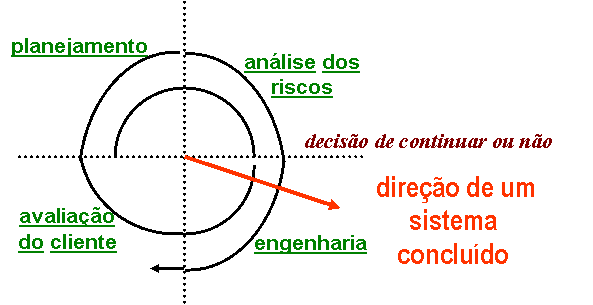
**direção de um sistema concluído**

**avaliação do cliente**

**engenharia**

**análise dos riscos**

**planejamento**



****

**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:** desenvolvimento do produto no nível seguinte
4. **Avaliação do Cliente:** avaliação do produto e planejamento das novas fases

* 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

**Comentário**

*O modelo é relativamente novo e não tem sido amplamente usado.*

*Demorará muitos anos até que a eficácia desse modelo possa ser determinada com certeza absoluta.*

**O Modelo Transformação (*Jair C Leite, 2000) Desenvolvimento formal***

Um programa é uma descrição formal, isto é, ele é descrito por uma linguagem de programação cuja sintaxe e semântica são definidos formalmente. Apenas desta forma é que temos a garantia de que o programa será sempre executado da mesma forma pelo computador.

O modelo Tranformação tem suas raízes na abordagem de métodos formais para o desenvolvimento de software. A idéia é que o desenvolvimento deve ser visto como uma seqüência de passos que gradualmente transforma uma especificação formal num programa. O passo inicial é transformar os requisitos informais numa especificação funcional formal. Esta descrição  formal é então transformada numa outra descrição formal mais detalhada, e assim, sucessivamente, até chegar no ponto em que se tenha componentes de programas que satisfaçam a especificação. Durante este processo de transformações sucessivas - chamado de *otimização* - as especifição formais produzidas podem ser executadas por um processador abstrato, permitindo uma verificação formal da sua validação antes mesmo que o programa venha a ser implementado. Antes de serem transformadas cada especificação formal deve ser verificada em relação às expectativas dos clientes e usuários.

As transformações devem ser realizadas pelo engenheiro de software. A natureza formal da transformação possibilitam a aplicação de verificações matemáticas como prova, consistência e completude da especificação. As transformações podem ser realizadas de maneira automática por ferramentas que traduzem especificações formais mais abstratas em especificações mais detalhadas.

Este modelo prevê que o engenheiro de software deve ter a sua disposição uma biblioteca de componentes reutilizáveis que satisfaça especificações formais. Na *otimização*, à medida que as especificações de mais baixo nível vão sendo obtidas verifica-se quais componentes da biblioteca implementam parte desta especificação. Um ambiente automatizado de desenvolvimento (uma ferramenta CASE - Computer Aided Software Engineering) pode auxiliar este processo armazenando o histórico de decisões dos engenheiros de software.

**Modelo Incremental**

|  |
| --- |
|  |
|  | http://inf.unisul.br/%7Evera/egs/aula01_arquivos/image006.gif |

 O modelo incremental segundo Pressman (2006) combina elementos do modelo cascata

sendo aplicado de maneira interativa. **O modelo de processo incremental é interativo**

**igual à prototipagem, mas diferente da prototipagem o incremental tem como objetivo apresentar um produto operacional a cada incremento realizado**.

Esse modelo é muito útil quando a empresa não possui mão de obra disponível no momento para uma implementação completa, dentro do prazo estipulado.

 A idéia principal deste modelo, é a de que um sistema deve ser desenvolvido de forma incremental, sendo que cada incremento vai adicionando ao sistema novas capacidades funcionais, até a obtenção do sistema final, sendo que, a cada passo realizado, modificações podem ser introduzidas.

* No primeiro passo deste modelo uma implementação inicial do sistema é obtida, na forma de um subconjunto da solução do problema global.
* criação de uma lista de controle de projeto, a qual deve apresentar todos os passos a serem realizados para a obtenção do sistema final.
* cada iteração do modelo de Desenvolvimento Iterativo consiste em retirar um passo da lista de controle de projeto através da realização de três etapas: o projeto, a implementação e a análise.
* O processo avança, sendo que a cada etapa de avaliação, um passo é retirado da lista, até que a lista esteja completamente vazia
* A lista de controle de projeto gerencia todo o desenvolvimento, definindo quais tarefas devem ser realizadas a cada iteração
* as tarefas na lista podem representar, inclusive, redefinições de componentes já implementados, em razão de erros ou problemas detectados numa eventual etapa de análise.
* a facilidade em testar o sistema, uma vez que a realização de testes em cada nível de desenvolvimento é, sem dúvida, mais fácil do que testar o sistema final.
* como na Prototipação, a obtenção de um sistema, mesmo incompleto num dado nível, pode oferecer ao cliente interessantes informações que sirvam de subsídio para a melhor definição de futuros requisitos do sistema.

**Desenvolvimento Rápido de aplicações (RAD) [**[**http://www.eps.ufsc.br/disc/procuml/#material**](http://www.eps.ufsc.br/disc/procuml/#material)**-Professores:** [**Marcello Thiry**](http://www.sj.univali.br/%7Ethiry) **e** [**José Salm Jr**](http://www.eps.ufsc.br/%7Esalm)**.]**

Não é exatamente um modelo e se baseia em:

– que um modelo de ciclo de vida formal é ineficiente e muitas revisões e documentações geradas pelos modelos em cascata e em espiral são perda de tempo

– a formalidade dificulta a comunicação com o cliente

– não há um modelo de ciclo de vida bem definido: há uma seqüência de integrações evolucionárias ou protótipos que são revisados com o cliente (os requisitos são levantados a partir destas iterações)

– cada integração está associada a um determinado

**Passos do RAD**

**1. Analisar requisitos;**

**2. Desenvolver projeto inicial;**

**3. Faça até estar pronto**

**{**

**Desenvolver a versão dentro do**

**tempo;**

**Entregar release para o cliente;**

**Receber o feedback;**

**Planejar versão para responder ao**

**feedback;**

}

**Características do RAD**

• Separação entre fases e atividades: cada time box inclui análise, projeto e implementação.

• Integrações constantes.

• Ênfase na codificação e não na especificação.

• Enfoque na reutilização.

• Envolvimento contínuo do cliente.

**Atividades para Resolver um Problema**

• Definição do escopo: entender o problema.

• Projeto: desenvolver uma abordagem para resolver o problema (usualmente usando

algum tipo de diagrama).

• Implementação: executar o projeto.

• Verificação: confirmar se a solução resolve o problema original.

**Ferramentas CASE - 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

**Ferramentas do ambiente de desenvolvimento de software de 4GL**

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*

**Engenharia de Software uma visão genérica**

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

**1. DEFINIÇÃO,**

**2. DESENVOLVIMENTO** e

**3. MANUTENÇÃO.**

**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.

**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 algorítmicos e as características de interface.

**MANUTENÇÃO:** concentra-se nas **“mudanças”** que ocorrerão depois que o software for liberado para uso operacional

***Correção***

***Adaptação***

***Melhoramento Funcional***

***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.

***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.

**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.

**.**