

## 2ª aula: Diferenças e Perfil do Analista; Definição e Ciclo de Vida de Sistemas

### Analistas X Programadores

Qualquer um que vive na área já deve ter visto o clássico ciclo de vida do desenvolvimento de sistemas. Salvo pequenas variações para cada modelo ou peculiaridades de cada projeto, o tal ciclo seria composto do estudo de viabilidade, análise, modelagem, implementação e implantação. Há quem considere a fase de testes uma fase independente, o que não faz grande diferença prática.

Se não há grandes divergências quanto ao ciclo em si, o caro leitor já deve estar curioso para saber do que trata este texto ou está tentado a abandonar a leitura por aqui. Antes que isto ocorra, esclareço: o objeto deste texto é a causa da briga entre analistas e programadores: o tamanho e importância das fases. São basicamente quatro perguntas :

- Quando começar a implementação, ou seja a codificar ?
- Pode a implementação concorrer no tempo com a análise ? E com a modelagem ?
- Quando termina a fase de modelagem ?
- A mesma pessoa que fez a análise pode fazer a codificação ?

### Os puristas

Para os engenheiros de software puristas clássicos, a alma do sistema está definida na fase de análise e o corpo na fase de modelagem. A implementação não envolve grande inteligência, é como se o sistema já estivesse pronto e a implementação fosse apenas o girar do interruptor para a posição ON. Por mais que esteja crescendo, este grupo ainda é bem pequeno, a grande maioria dos sistemas desenvolvidos já são iniciados pelo código, documentação são alguns comentários esparsos pelos fontes e análise é o conjunto de rabiscos em que resultou a reunião da equipe. O analista purista domina as ferramentas CASE, metodologias, mas normalmente não se aventura a escrever uma linha de código sequer.

### Visual Drag and Drop

Há alguns anos desenvolvimento de sistemas era uma coisa absolutamente desconhecida, atividade restrita a um seleto grupo de gênios, nerds ou malucos, simplesmente. A maioria das ferramentas de desenvolvimento era interpretada ou levava horas para compilar um programa de tamanho médio. Ainda no início dos anos 90 era comum os desenvolvedores compilarem seus programas durante toda a noite, para fazer os testes no dia seguinte. Com toda esta limitação técnica, era fundamental um mínimo de análise e teste antes de ser digitada a primeira linha de código, pois um simples erro de sintaxe poderia custar um dia inteiro de trabalho.

Hoje os tempos são outros, ao mesmo tempo que as máquinas evoluíram a ponto de diminuir a importância de parte da atividade pré- codificação, como o famoso teste de mesa, as ferramentas de desenvolvimentos evoluíram para interfaces visuais e amigáveis. O resultado é que quase todo mundo hoje é capaz de desenvolver sistemas. Esses dois fatores combinados resulta numa desvalorização da fase de análise.

Uma pessoa é capaz de criar algo funcional usando algo como o Visual Basic, sem sequer imaginar o que seja objeto, classe, método ou mensagem. Mas não só os leigos são adeptos do *Visual Drag and Drop, Just do it*, muitos bons profissionais da área abandonaram grande parte do seu conhecimento teórico e também adotaram a metodologia *Just in Time* de desenvolvimento. Toda esta gente é tecnicamente um programador de sistemas, surge até o que tem-se chamado de Analista Programador, na verdade um programador com salário mais alto.

## O Confronto

Tudo correria bem, cada grupo e suas convicções trabalhariam tranqüilamente, se não houvesse o encontro na mesma equipe de membros dos dois grupos. Tudo transcorre bem no início, nas fases de estudo de viabilidade e de requisitos. As divergências aparecem quando a fase de projeto começa a se alongar e os programadores começam a ficar ansiosos para codificar. Normalmente o que se faz é buscar tarefas periféricas para ir distraindo o programador até que os analistas consigam adiantar o projeto até um ponto satisfatório.

## Os argumentos

Tanto analistas clássicos, quanto os programadores têm argumentação forte para defender seu ponto de vista.

O analista:

- No desenvolvimento sem análise, gasta-se cerca de 80% do tempo refazendo código e corrigindo bugs. A troca de um membro da equipe é muito mais traumática.

O programador:

- O cliente paga pelo programa rodando, pelo código. É muito mais complicado justificar o tempo gasto, sem código para ser mostrado. A análise não interessa para o cliente.

De fato ambos os argumentos são pertinentes, de modo que não há como definir de que lado está a razão. A grande questão é como administrar e tirar o máximo proveito de uma equipe mista.

## Boa convivência

Deixando de lado o radicalismo, há sempre uma forma de ter analistas e programadores em harmonia, fazendo com que as diferenças contribuam para o engrandecimento do grupo como um todo. Como fazer isto? A resposta passa pelas quatro questões iniciais:

- Quando começar a implementação, ou seja a codificar ?

Cada um tem de atuar onde é mais forte. Não adianta muito forçar um programador a fazer análise. Logo, até que haja código a ser escrito, os programadores estarão subaproveitados. Uma boa análise certamente decomporá o sistema em módulos com funcionalidade própria. Alguns módulos podem ser definidos quase que totalmente logo no início do projeto. Um exemplo usual é a comunicação como banco de dados, mas há diversas outras possibilidades. Numa equipe onde a utilização de recursos é otimizada, esses módulos serão projetados o quanto antes e ,enquanto são projetados os demais módulos, os programadores já estarão trabalhando na codificação dos módulos iniciais.

- Pode a implementação concorrer no tempo com a análise ? E com a modelagem ?

Não há nada que impeça que os programadores estejam codificando um módulo, enquanto os analistas sequer analisaram um outro módulo. Fazendo a analogia com as clássicas árvores de busca, basta que o desenvolvimento seja feito, pelo menos em parte, em profundidade e não em largura , como os analistas tendem a pensar de início.

- Quando termina a fase de modelagem ?

Há uma máxima na computação que diz que um sistema jamais estará realmente finalizado, sempre há o que alterar, corrigir ou melhorar. Um erro que muitos de nós cometemos é não manter a sincronização entre a análise, o modelo e a implementação. Muitas vezes é tão prático corrigir um bug ou fazer uma alteração, que sequer passa-nos pela cabeça voltarmos ao modelo. Quando estamos utilizando ao máximo os recursos que dispomos, o

modelo está ao lado da implementação e , ao contrário do que possa parecer, o trabalho dos analistas não terminará antes que o dos programadores.

- Quando termina a fase de modelagem ?

Esta é a pergunta mais fácil de responder: NUNCA. Se um sistema jamais está pronto e se o projeto deve estar sincronizado com a implementação, o projeto, da mesma forma, jamais estará concluído.

- A mesma pessoa que fez a análise pode fazer a codificação ?

Até pode, mas será que estaremos tirando o máximo dos recursos disponíveis? Evidentemente que não. Pelé era um bom goleiro, chegou até a atuar nesta posição algumas vezes, por quê então ele não jogava metade do tempo no gol e outra metade como ponta de lança ?

### SISTEMA DE INFORMAÇÃO

Em um Sistema, várias partes trabalham juntas visando um objetivo em comum. Em um Sistema de Informação não é diferente, porém o objetivo é um fluxo mais confiável e menos burocrático das informações. Em um Sistema de Informação bem construído, suas principais vantagens são: Otimização do fluxo de informação permitindo maior agilidade e organização; Redução de custos operacionais e administrativos e ganho de produtividade; Maior integridade e veracidade da informação; Maior estabilidade; Maior segurança de acesso à informação.

Informações de boa qualidade são essenciais para uma boa tomada de decisão.

Observações: Um Sistema de Informação não precisa ter essencialmente computadores envolvidos, basta ter várias partes trabalhando entre si para gerar informações.

Ele pode ser tanto manual quanto baseado em TI, ou uma mescla dos dois. Acontece que um Sistema de Informação grande dificilmente sobrevive atualmente sem estar informatizado, o que por si só não elimina o fator humano no processo. É a interação dos componentes da TI com o componente humano que faz com que um Sistema de Informação tenha funcionalidade e utilidade para a organização.

### Classificação

Podemos ter a classificação dos Sistemas de Informação baseados em TI de acordo com o tipo de informação processada:

- **Sistemas de Informação Operacional:** tratam das transações rotineiras da organização; Comumente encontrados em todas as empresas automatizadas
- **Sistemas de Informação Gerencial:** agrupam e sintetizam os dados das operações da organização para facilitar a tomada de decisão pelos gestores da organização;
- **Sistemas de Informação Estratégicos:** integram e sintetizam dados de fontes internas e externas à organização, utilizando ferramentas de análise e comparação complexas, simulação e outras facilidades para a tomada de decisão da cúpula estratégica da organização.

### CICLO DE VIDA DE UM SISTEMA

Os modelos existentes possuem diferentes graus de sofisticação e complexidade. Para projetos que envolvem uma equipe de desenvolvimento pouco numerosa e experiente, o mais adequado será provavelmente um processo simples. No entanto, para sistemas maiores que envolvem equipes de dezenas ou centenas de elementos e milhares de utilizadores, um processo simples não é suficiente para oferecer a estrutura de gestão e disciplina

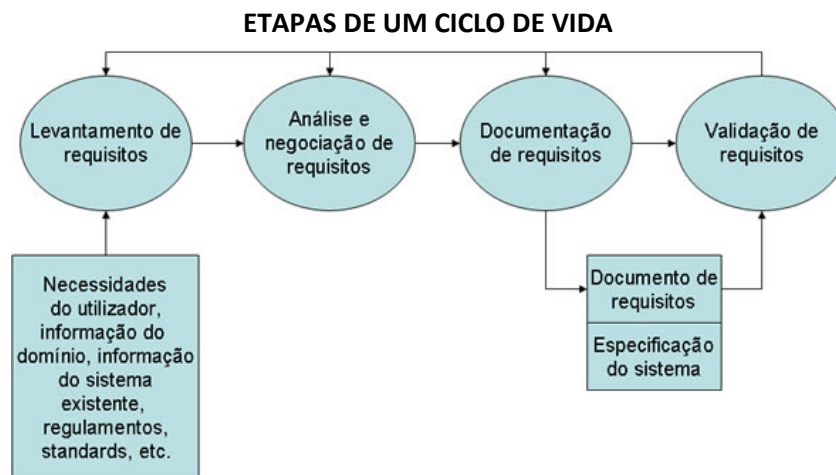
necessárias à engenharia de um bom produto de software. Desta forma, é necessário algo mais formal e disciplinado. É importante fazer notar que isto não significa que se perca em inovação ou que se põe entraves à criatividade. Significa apenas que é utilizado um processo bem estruturado para permitir a criação de uma base estável para a criatividade.

Por mais simples ou complexo que possa parecer, um modelo de ciclo de vida de um projeto é, de fato, uma versão simplificada da realidade. É suposto ser uma abstração e, tal como todas as boas abstrações, apenas a quantidade de detalhe necessária ao trabalho em mãos deve ser incluída. Qualquer organização que deseje por um modelo de ciclo de vida em prática irá necessitar de adicionar detalhes específicos para dadas circunstâncias e diferentes culturas. Por exemplo, a Microsoft quis manter uma cultura de pequena equipa e ao mesmo tempo tornar possível o desenvolvimento de grandes e complexos produtos de software.

### Modelos em Engenharia de Software e Requisitos

A engenharia de software tem produzido inúmeros modelos de ciclo de vida, incluindo os modelos de cascata, espiral e desenvolvimento rápido de aplicações (RAD). Antes do modelo de cascata ser proposto em 1970, não havia concordância em termos dos métodos a levar a cabo no desenvolvimento de software. Desde então ao longo dos anos muitos modelos têm sido propostos refletindo assim a grande variedade de interpretações e caminhos que podem ser tomadas no desenvolvimento de software. Neste artigo, foi decidida a inclusão destes modelos por duas razões: primeiro porque são representativos dos modelos utilizados na indústria e foi já provado o seu sucesso, e segundo porque mostram como a ênfase no desenvolvimento de software mudou gradualmente de forma a incluir uma visão mais interativa e centrada no utilizador.

CICLO DE VIDA	VANTAGENS	DESVANTAGENS
<b>Modelo em FASE</b>	<ul style="list-style-type: none"> <li><input type="checkbox"/> Fortemente documentado</li> <li><input type="checkbox"/> Visa Alta Qualidade</li> <li><input type="checkbox"/> Enfatiza metas e pontos de revisão</li> </ul>	<ul style="list-style-type: none"> <li><input type="checkbox"/> Improdutivo quanto ao tempo</li> <li><input type="checkbox"/> Sua visão Seqüencial não corresponde ao mundo real</li> </ul>
<b>Modelo em Protótipo</b>	<ul style="list-style-type: none"> <li><input type="checkbox"/> Facilidade de percepção por parte do usuário</li> <li><input type="checkbox"/> Não exige grande quantidade de detalhamento</li> </ul>	<ul style="list-style-type: none"> <li><input type="checkbox"/> O risco do protótipo passar o sistema em produção</li> <li><input type="checkbox"/> No descarte do protótipo pode se perder as especificações de requisitos.</li> </ul>
<b>Modelo em Espiral</b>	<ul style="list-style-type: none"> <li><input type="checkbox"/> Permite a resolução do sistema por partes</li> <li><input type="checkbox"/> Melhora o tempo de implementação do sistema</li> </ul>	<ul style="list-style-type: none"> <li><input type="checkbox"/> Continua dando mais ênfase a parte funcional</li> </ul>
<b>Modelo Automatizado</b>	<ul style="list-style-type: none"> <li><input type="checkbox"/> Manutenção da especificação de requisitos ao invés de código</li> <li><input type="checkbox"/> Maior facilidade de interação por parte do usuário</li> <li><input type="checkbox"/> Permite criar facilmente protótipos</li> </ul>	<ul style="list-style-type: none"> <li><input type="checkbox"/> Necessidade de um conversor automático</li> <li><input type="checkbox"/> Seu nível de abstração depende do conversão</li> </ul>



### Análise de Requisitos

Esta fase visa identificar o tipo de serviço de processamento de dados a ser executado (manutenção de um software existente ou a criação de um outro), os objetivos a serem alcançados, recursos e prazos necessários para a execução do projeto.

O Resultado é um documento denominado Anteprojeto, contendo o modelo lógico preliminar do software. A aprovação deste documento pelo usuário torna-se pré-requisito para a continuidade do trabalho.

### Projeto Lógico

Nesta fase o objetivo é a especificação detalhada dos elementos do software a nível lógico. Além disso, deve tratar da especificação detalhada dos procedimentos externos ao computador, tais como:

- Captação das informações;
- Preparo e envio para processamento;
- Crítica e correções;
- Distribuição das saídas.

O produto é um documento denominado Manual do Software - Parte I - Projeto Lógico, que deverá ser submetido ao usuário para análise e aprovação.

### Projeto Físico

Tendo como base o Projeto Lógico, o objetivo nesta fase é o de detalhar os elementos do software a nível físico.

O Produto é um documento denominado Manual do Software - Parte II - Projeto Físico, que conterà a especificação técnica completa do software, visando a sua implementação.

### Implementação

O objetivo desta fase é o desenvolvimento e simulação do software especificado no Projeto Físico. O resultado são os programas fontes, devidamente testados. Estes, por sua vez, devem ser entregues ao usuário via disquetes.

### Implantação

Tem como objetivo o treinamento do usuário, a conversão/inicialização de arquivos e a implantação do software para produção.

Nesta fase, é elaborado e entregue o Manual do Usuário, assim como o Termo de Encerramento do Desenvolvimento do Software, onde o analista ou empresa desenvolvedora declara que o software, uma vez implantado, está entregue e considerado, aceito: devendo o mesmo entrar no período de garantia.

### **Operação**

Nesta Fase são executadas as atividades de produção do software pelo usuário, com acompanhamento inicial da execução das rotinas, avaliação da performance, pequenos ajustes e análise de resultados.

O produto é um relatório descritivo dos problemas encontrados pelo usuário e as soluções adotadas, e a documentação do software, como um todo, devidamente revisada.

### **Anteprojeto (Análise de Requisitos)**

As atividades executadas nesta fase são as seguintes:

#### **Identificação dos Objetivos**

Em função das necessidades apresentadas pelo usuário e do tipo de serviço a ser executado (manutenção ou desenvolvimento), identificar o objetivo global e os específicos do software.

#### **Definição da Abrangência**

- Em conjunto com o usuário e com base nos objetivos:
- Identificar as macro funções existentes, os órgãos envolvidos, as pessoas responsáveis por esses e nome dos participantes para contatos.
  - Descrever sucintamente os objetivos das macros funções envolvidas.
  - Elaborar o DFD de contexto, mostrando os fluxos de dados recebidos e os originados e, suas respectivas origem(s)/destino(s). As origens/destinos identificam-se com softwares, unidade organizacional, pessoas, organizações externas, etc.

#### **Análise de Dados**

Identificar, junto aos usuários, as principais Entidades, Atributos e Relacionamentos. Neste ponto, deve-se elaborar o Modelo Lógico de Dados(DER) e correspondente lista de entidades.

#### **Análise de Problemas**

Com base nos objetivos das macros funções, identificar junto aos responsáveis pelas mesmas, os problemas existentes, suas causas, seus efeitos e ação para a solução dos mesmos.

#### **Situação Pretendida**

Definir a situação pretendida buscando atender às necessidades estabelecidas pelo usuário, bem como a eliminação dos problemas existentes.

#### **Fluxo dos Dados**

Identificar, num primeiro particionamento do DFD de contexto, os fluxos de dados de entrada e saída de cada macro função, assim como os depósitos de dados envolvidos, gerando um ou mais níveis de DFD de acordo com a necessidade de análise desta fase.

Os depósitos de dados representados no DFD devem espelhar as entidades do Modelo Lógico de Dados (DER) (Depósito de dados = Entidade não Normalizada).

#### **Documentação Atual**

Relacionar/reunir cópia(modelo) de documentos e relatórios utilizados, para efeitos de orientação.

#### **Alternativas de Hardware e Software de Apoio**

Deverão ser procuradas alternativas de hardware e de software de apoio. Em cada uma delas, deverá ser feita uma análise dos benefícios em conjunto com o usuário, devendo ser escolhida como a solução proposta aquela que apresentar melhores vantagens. A solução deverá atender tanto a aspectos de desenvolvimento e de operação.

#### **Estimativas de Recursos Humanos e Prazos**

Identificar recursos humanos e respectivos prazos necessários ao desenvolvimento e implantação do software proposto, contemplando inclusive atividades tais como: tarefas de conversão, de treinamento, de documentação e outros.

#### **Controle de Qualidade da Fase**

Tendo como referência os critérios para revisão da análise estruturada, realizar a referida revisão prevista para esta fase. Inclusive verificando a adequação do documento àquele determinado pela metodologia. Avaliar a solução proposta em termos técnicos, recursos físicos e financeiros, assim como o prazo de execução.

### **EXEMPLOS DE APLICAÇÃO**

#### Objetivo

- Automatiza e padroniza a documentação dos sistemas.
- Aumenta a produtividade no desenvolvimento de sistema.
- Melhora a qualidade do software.
- Aumenta a interação do usuário na definição de sistemas.

#### Benefícios

- Acelera o processo de criação do software
- Permite o processo de abordagem.
- Facilita a manutenção.
- Assegura maior aderência os requisitos.
- Facilita a atuação gerencial.

#### Tendências

- Tecnologia case substituindo linguagens de 4° geração.
- Necessidade de treinamento e consultoria continua.
- Casamento entre o Case, VB, PB, Delphi e outras.
- Mercado esta em transição.
- Ambientes 100 % integrados e competentes.
- Métodos cada vez mais desenvolvidos

#### Características Técnicas

- Suporte e metodologia de dados e estruturas
- Integração entre as fases analise, projeto aplicação.
- Geração de código para linguagens
- Engenharia de reservas.

### Características Genéricas

- Documentação on-line e manual.
- Tutorias e exemplos.
- Facilidade de instalação
- Gerador de relatórios.
- Suporte técnico.
- Custo por copia.
- Posição no mercado brasileiro

### Características Estruturais

- Multiusuário.
- Facilidade de utilização.
- Reposito compartilhado.
- Controle de versões.
- Interfase GUI.
- Suporte a OLE.

### Linhas de Desenvolvimento

- Tecnologia de reposito de dados.
- Aceleração do processo de desenvolvimento.
- Portabilidade e Interabilidade.
- Métodos.

### Funcionalidades Básicas

- Diagrama modelo específico informações de dados no sistema.
- Verifica a integridade de diagrama de dados.
- Disponibiliza técnicas de modelagem.
- Automatiza a padroniza o processo de documentação dos sistemas.
- Gera aplicativos para banco de dados e *front ends*.

### Característica Genérica das Ferramentas CASE

- Direcionadas os principais problemas de construção de software.
- Lida meta modelos e dados.
- Integra informações de diferentes plataformas.
- Linguagens de 3° e 4° geração.
- Bancos de dados relacionados.
- Ferramentas OO
- Adoção cara e lenta: Treinamento, consultoria, licenças.
- Poucos departamentos de IT adotam métodos organizados para a construção de software.

### Análise das Ferramentas de Mercado

- Ambiente de desenvolvimento diversificado.
- Windows 95 e NT.
- VB, Delphi, Powerbuilder.
- Oracle, SQL Server, Sybase, Informix
- Presença no mercado brasileiro.
- Disponibilização da ferramenta e de material de consulta técnica.

### Cenário Atual Promessa ou Realidade

- Foram badaladas no início dos 80.
- Mercado estimado em U\$ 80 milhões de (90 s).
- Taxa de crescimento de 10 %.
- Outro mercado de IT tem 30 % em média ou mais.
- Hoje aproximadamente U\$ 1,5 bilhões.

### Situação Atual das Ferramentas de CASE

- São componentes apenas adicionais do ambiente de desenvolvimento.
- Presença inexpressiva em pequenas organizações de IT.
- Geração de código ineficiente.
- Suporte não acompanha o desenvolvimento do projeto lógico para o físico.

### Maturidade das Ferramentas de CASE

- Termo genérico usado de forma indiscriminada.
- Ambiente de desenvolvimento.
- Ferramenta de desenvolvimento.
- CASE simplesmente.
- Inclusão de novos componente (geração de códigos)

### Dificuldade em Implantar CASE

- Oposto da tecnologia “plug and play”.
- Necessidade de treinamento intensivo e apoio de consultoria.
- São geralmente empregadas em problemas de difícil solução.
- Os problemas de sistemas legados e uma das principais aplicações de CASE.

### Então Por que Ferramenta Case?

Significa um avanço radical em todas as etapas do processo de construção de software.

Qualidade em sistemas ISO 9001/ 3      CMM( Capability Maturity Model)

**TRABALHO: Visão Geral sobre CICLOS DE VIDA DE SOFTWARE**

**Entrega dia 17/03**