



## **Análise de Sistemas**

### **Aula 1**

Prof. Emerson Klisiewicz

## **Contextualização**

### **Apresentação**

- Formação acadêmica
  - Especialista em Redes e Sistemas Distribuídos
  - Especialista em Sistemas de Informações Gerenciais
  - Bacharel em Ciência da Computação

- Atividades profissionais
  - SPEI
  - FACET
  - HSBC

### **Conteúdo da Disciplina**

- Aula 1
  - Introdução à Análise de Sistemas
  - Crise do *Software*
  - Metodologias Clássicas

- Aula 2
  - Análise Essencial
  - Análise Estruturada
  - Análise Orientada a Objetos
- Aula 3
  - Engenharia de *Software*
  - Engenharia de Requisitos
  - Requisitos e Tipos de Requisitos



- Aula 4
  - Gerenciamento dos Requisitos
  - Refinamento de Requisitos
  - Aprovação de Requisitos
  - Matriz de Rastreabilidade
- Aula 5
  - Análise Orientada a Objetos

- Introdução a UML – Histórico e Visão Geral
- Ferramentas CASE para a UML
- Aula 6
  - Diagrama de Casos de Uso
  - Diagrama de Classes e Diagrama de Objetos
  - Diagrama de Sequência
  - Diagrama de Máquina de Estados

### **Aula 1**

- Introdução à Análise de Sistemas
- Crise do *Software*
- Ciclo do *Software*

### **Introdução**

- As grandes transformações ocorridas nos últimos anos, impulsionadas pelo avanço da tecnologia provocaram a passagem da antiga sociedade industrial para uma nova sociedade baseada na informação e no conhecimento

- Nos dias de hoje, a empresa que dispõe de mais informações sobre seu processo está em vantagem em relação a suas competidoras

### **Instrumentalização**



## Um Pouco de História

### Evolução

- Década de 1950/60: os sistemas de *software* eram bastante simples e dessa forma as técnicas de modelagem também
- Era a época dos fluxogramas e diagramas de módulos

- Década de 1970: nessa época houve uma grande expansão do mercado computacional. Sistemas complexos começavam a surgir e, por consequência, modelos mais robustos foram propostos. Nesse período surge a programação estruturada e no final da década a análise e o projeto estruturado

- Década de 1980: surge a necessidade de interfaces homem-máquina mais sofisticadas, o que originou a produção de sistemas de *software* mais complexos

- A análise estruturada se consolidou na primeira metade dessa década e em 1989 Edward Yourdon lança o livro *Análise Estruturada Moderna*, tornando-o uma referência no assunto

- Década de 1990: nesse período surge um novo paradigma de modelagem, a Análise Orientada a Objetos, como resposta a dificuldades encontradas na aplicação da Análise Estruturada a certos domínios de aplicação



- Final da década de 1990 e momento atual: o paradigma da orientação a objetos atinge a sua maturidade. Os conceitos de padrões de projetos (*design patterns*), *frameworks* de desenvolvimento, componentes e padrões de qualidade começam a ganhar espaço

- Nesse período surge a Linguagem de Modelagem Unificada (UML), que é a ferramenta de modelagem utilizada no desenvolvimento atual de sistemas

### **Mas o que é *software*?**

- 1. Instruções** que quando executadas produzem a função e o desempenho desejados

- 2. Estruturas de dados** que possibilitam que os programas manipulem adequadamente a informação
- 3. Documentos** que descrevem a operação e o uso dos programas

### **Características do *Software***

1. Desenvolvido ou projetado por engenharia, não manufaturado no sentido clássico

2. Não se desgasta, mas se deteriora
3. A maioria é feita sob medida em vez de ser montada a partir de componentes existentes



## **Crise de *Software***

- Refere-se a um conjunto de problemas encontrados no desenvolvimento de *software*:

### **1. As estimativas de prazo e de custo frequentemente são imprecisas**

- “Não dedicamos tempo para coletar dados sobre o processo de desenvolvimento de *software*.”
- “Sem nenhuma indicação sólida de produtividade, não podemos avaliar com precisão a eficácia de novas ferramentas, métodos ou padrões.”

### **2. A produtividade das pessoas da área de *software* não tem acompanhado a demanda por seus serviços**

- “Os projetos de desenvolvimento de *software* normalmente são efetuados apenas com um vago indício das exigências do cliente.”

### **3. A qualidade de *software* às vezes é menos que adequada**

- Só recentemente começam a surgir conceitos quantitativos sólidos de garantia de qualidade de *software*

### **4. O *software* existente é muito difícil de manter**

- A tarefa de manutenção devora o orçamento destinado ao *software*
- A facilidade de manutenção não foi enfatizada como um critério importante



### Crise de *Software* – Resumindo

- Estimativas de prazo e de custo ↑
- Produtividade das pessoas ↓
- Qualidade de *software* ↓
- *Software* difícil de manter ↑

### Causas dos Problemas Associados à Crise de *Software*

#### 1. Próprio caráter do *software*

- O *software* é um elemento de sistema lógico e não físico

- Consequentemente o sucesso é medido pela qualidade de **uma única entidade** e não pela qualidade de muitas entidades manufaturadas
- **O *software* não se desgasta, mas se deteriora**

#### 2. Falhas das pessoas responsáveis pelo desenvolvimento de *software*

- Gerentes sem nenhum *background* em *software*

- Os profissionais da área de *software* têm recebido pouco treinamento formal em novas técnicas para o desenvolvimento de *software*
- Resistência a mudanças

#### 3. Mitos do *software*

- Propagaram desinformação e confusão

- ✓ **Administrativos**
- ✓ **Cliente**
- ✓ **Profissional**





### Mitos do *Software* (Administrativos)

- **Mito:** Já temos um manual repleto de padrões e procedimentos para a construção de *software*. Isso não oferecerá ao meu pessoal tudo o que eles precisam saber?



#### ▪ **Realidade:**

- Será que o manual é usado?
- Os profissionais sabem que ele existe?
- Ele reflete a prática moderna de desenvolvimento de *software*? Ele é completo?

- **Mito:** Meu pessoal tem ferramentas de desenvolvimento de *software* de última geração, afinal compramos os mais novos computadores

- **Realidade:** É preciso muito mais do que os mais recentes computadores para se fazer um desenvolvimento de *software* de alta qualidade

- **Mito:** Se nós estamos atrasados nos prazos, podemos adicionar mais programadores e tirar o atraso

- **Realidade:** O desenvolvimento de *software* não é um processo mecânico igual à manufatura. Acrescentar pessoas em um projeto torna-o ainda mais atrasado
- Pessoas podem ser acrescentadas, mas com critérios. Somente de uma forma planejada



### Mitos do *Software* (Cliente)

- **Mito:** Uma declaração geral dos objetivos é suficiente para se começar a escrever programas – podemos preencher os detalhes mais tarde

- **Realidade:** Uma definição inicial ruim é a principal causa de fracassos dos esforços de desenvolvimento de *software*. É fundamental uma descrição formal e detalhada do domínio da informação, função, desempenho, interfaces, restrições de projeto e critérios de validação

- **Mito:** Os requisitos de projeto modificam-se continuamente, mas as mudanças podem ser facilmente acomodadas, porque o *software* é flexível

- **Realidade:** Uma mudança, quando solicitada tardiamente num projeto, pode ser maior do que a ordem de magnitude mais dispendiosa da mesma mudança solicitada nas fases iniciais

### Mitos do *Software* (Profissional)

- **Mito:** Assim que escrevermos o programa e o colocarmos em funcionamento nosso trabalho estará completo

- **Realidade:** Os dados da indústria indicam que entre 50 e 70% de todo esforço gasto num programa serão despendidos **depois** que ele for entregue pela primeira vez ao cliente





- **Mito:** Enquanto não tiver o programa “funcionando”, eu não terei realmente nenhuma maneira de avaliar sua qualidade

- **Realidade:** Um programa funcionando é somente uma parte de uma configuração de *software* que inclui todos os itens de informação produzidos durante a construção e manutenção do *software*
  - Ex.: Avião

### **Análise de Sistemas**

- Estudo da organização e funcionamento de uma ou mais atividades, com o objetivo de gerar um conjunto de ações informatizada que solucione, da melhor forma possível, um problema ou automatize uma ação manual

- **Métodos:** proporcionam os detalhes de como fazer para construir o *software*
  - Planejamento e estimativa de projeto
  - Análise de requisitos de *software* e de sistemas

- Projeto da estrutura de dados
- Algoritmo de processamento
- Codificação
- Teste
- Manutenção

- **Ferramentas:** dão suporte automatizado aos métodos
- Existem atualmente ferramentas para sustentar cada um dos métodos



- Quando as ferramentas são integradas é estabelecido um sistema de suporte ao desenvolvimento de *software* chamado *CASE – Computer Aided Software Engineering*

- **Procedimentos:** constituem o elo de ligação entre os métodos e ferramentas
- Sequência em que os métodos serão aplicados
- Produtos que se exige que sejam entregues

- Controles que ajudam a assegurar a qualidade e coordenar as alterações
- Marcos de referência que possibilitam administrar o progresso do *software*

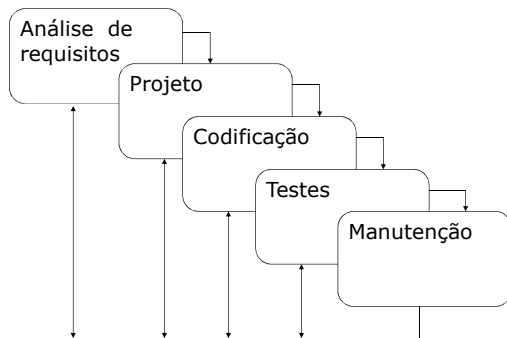
### **Análise de sistemas**

- Conjunto de etapas que envolve **métodos, ferramentas e procedimentos**

- Essas etapas são conhecidas como componentes de **ciclos de vida de *software***
- Alguns ciclos de vida mais conhecidos são: **ciclo de vida clássico, prototipação, modelo espiral**

### **Ciclo de Vida Clássico (Cascata)**

- Modelo mais antigo e o mais amplamente usado da engenharia de *software*
- Requer uma abordagem sistemática, sequencial ao desenvolvimento de *software*



## Atividades do Ciclo de Vida Clássico

### 1. 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

### 2. Projeto

- 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
  - ✓ Caracterização de Interfaces

### 3. Codificação

- Tradução das representações do projeto para uma linguagem "artificial" resultando em instruções executáveis pelo computador



#### 4. 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

#### 5. Manutenção

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

#### 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



- Adequado para quando o cliente definiu um conjunto de objetivos gerais para o *software*, mas não identificou requisitos de entrada, processamento e saída com detalhes

## 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)

- 3. Construção do protótipo:** implementação do projeto rápido

- 4. Avaliação do protótipo:**  
cliente e desenvolvedor avaliam o protótipo

- 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 de produto:**

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

**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
- Pode usar a **Prototipação**, em qualquer etapa da evolução do produto, como mecanismo de redução de riscos

**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

**Aplicação**



### **Ciclo de Vida Clássico**

- Projetos reais raramente seguem o fluxo sequencial que o modelo propõe

- Logo no início é difícil estabelecer explicitamente todos os requisitos. No começo dos projetos sempre existe uma incerteza natural
- O cliente deve ter paciência. Uma versão executável do *software* só fica disponível numa etapa avançada do desenvolvimento

### **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 ideia que a versão final do *software* vai ser construída e “força” a utilização do protótipo como produto final

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

- Depois de um tempo ele familiariza com essas escolhas, e esquece que elas não são apropriadas para o produto final



### **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

### **Síntese**

### **Resumindo**

- Introdução à Análise de Sistemas
- Crise do *software*
- Ciclo do *software*