Resumo Engenharia de Software (Aulas 01-09) - Baseado em Herysson/Engenharia-de-Software

---

Aula 01: Introdução à Engenharia de Software (ES)

- **O que é ES?** Disciplina da engenharia focada em todos os aspectos da produção de software, desde a concepção até a manutenção, buscando qualidade, custo e prazo adequados.
- **Software:** Programas + Documentação + Dados de configuração.
- **Crise do Software:** Dificuldades históricas (custo, prazo, qualidade) no desenvolvimento de sistemas complexos. ES surge como resposta.
- **Diferença ES vs Ciência da Computação:** ES = Foco na prática e construção de sistemas; CC = Foco nos fundamentos teóricos.
- **Qualidade de Software:** Atributos como funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade, portabilidade.
- **Ética e Profissionalismo:** Importância do comportamento ético na prática da ES.

Aula 02: Ciclo de Vida e Modelos de Processo de Software

- **Processo de Software:** Conjunto de atividades e resultados associados para produzir um produto de software.
- **Ciclo de Vida:** Fases fundamentais: Especificação -> Projeto (Design) -> Implementação -> Validação (Testes) -> Evolução (Manutenção).
- **Modelos de Processo:** Estruturas para organizar as atividades do ciclo de vida.
    - **Cascata (Waterfall):** Sequencial, fases distintas e lineares. Pouca flexibilidade.
    - **Prototipagem:** Criação de protótipos para validar requisitos com o cliente. Iterativo.
    - **Espiral (Spiral):** Iterativo e baseado em riscos. Combina prototipagem e cascata. Complexo.
    - **Incremental:** Entrega o software em partes funcionais (incrementos).
    - **RUP (Rational Unified Process):** Iterativo e incremental, focado em casos de uso e arquitetura. Disciplinado.
    - **(Modelos Ágeis - introdução):** Abordagens iterativas e adaptativas (serão detalhadas depois).

Aula 03: Princípios e Métodos de Desenvolvimento

- **Princípios Fundamentais da ES:**
    - **Rigor e Formalidade:** Abordagem sistemática.
    - **Separação de Interesses (Concerns):** Dividir problemas complexos em partes menores.

    - **Modularidade:** Componentes independentes e coesos.
    - **Abstração:** Focar no essencial, esconder detalhes.
    - **Antecipação a Mudanças:** Projetar para facilitar evoluções.
    - **Generalidade:** Criar soluções reutilizáveis.
    - **Incrementalidade:** Desenvolver em partes.
- **Métodos de Desenvolvimento:**
    - **Estruturado:** Foco em funções e fluxo de dados (DFDs). Abordagem mais antiga.
    - **Orientado a Objetos (OO):** Foco em classes e objetos, encapsulamento, herança, polimorfismo. Usa UML.
    - **Ágil:** Foco em colaboração, resposta rápida a mudanças, entregas frequentes (Ex: Scrum, XP).

Aula 04: Princípios de Modelagem de Software

- **O que é Modelagem?** Criar representações simplificadas (modelos) de um sistema ou de parte dele.
- **Por que Modelar?** Para entender, comunicar, analisar e documentar o sistema. Ajuda a lidar com a complexidade.
- **Princípios da Modelagem:**
    - Escolher modelos adequados ao problema.
    - Diferentes modelos revelam diferentes aspectos.
    - Nível de precisão depende do objetivo.
    - Modelos devem estar conectados à realidade.
- **UML (Unified Modeling Language):** Linguagem padrão para visualização, especificação, construção e documentação de artefatos de software. Não é um método, mas uma notação.
- **Tipos de Modelos:**
    - **Estruturais:** Mostram a estrutura estática (Ex: Diagrama de Classes).
    - **Comportamentais:** Mostram a dinâmica do sistema (Ex: Diagrama de Casos de Uso, Sequência, Atividade).

Aula 05: Requisitos de Software

- **O que são Requisitos?** Descrições do que o sistema deve fazer (serviços) e das restrições sob as quais ele deve operar.
- **Tipos de Requisitos:**
    - **Funcionais:** Declaram funcionalidades ou serviços. O *que* o sistema faz. (Ex: "O sistema deve permitir cadastrar cliente").
    - **Não Funcionais (RNF):** Restrições sobre os serviços ou qualidades do sistema. O *como* o sistema faz. (Ex: Performance, segurança, usabilidade, confiabilidade).
    - **De Domínio:** Vêm do domínio da aplicação, podem ser funcionais ou não. (Ex: Uma lei específica da área).

- **Engenharia de Requisitos (ER):** Processo de descobrir, analisar, documentar, validar e gerenciar requisitos.
    - **Fases:** Elicitação -> Análise -> Especificação -> Validação -> Gerenciamento.
- **Características de Bons Requisitos:** Completos, Consistentes, Corretos, Realizáveis, Verificáveis, Rastreáveis, Não ambíguos, Modificáveis.

Aula 06: Casos de Uso (Use Cases)

- **O que são Casos de Uso?** Técnica para capturar requisitos funcionais descrevendo interações entre atores (usuários ou sistemas externos) e o sistema para alcançar um objetivo específico.
- **Elementos Principais:**
    - **Ator:** Quem ou o que interage com o sistema.
    - **Caso de Uso:** Uma funcionalidade/objetivo específico (verbo + objeto, ex: "Cadastrar Cliente").
    - **Sistema:** Fronteira que delimita o software.
    - **Relacionamentos:** Include (inclusão obrigatória), Extend (extensão opcional), Generalização (herança entre atores ou casos de uso).
- **Diagrama de Casos de Uso (UML):** Representação gráfica das interações entre atores e casos de uso. Visão geral.
- **Especificação de Caso de Uso (Textual):** Descrição detalhada de um caso de uso: Nome, Atores, Pré-condições, Pós-condições, Fluxo Principal (caminho feliz), Fluxos Alternativos (exceções, variações).

Aula 07: Modelagem e Projeto de Software (Design)

- **O que é Projeto (Design)?** Processo de definir a arquitetura, componentes, interfaces e outros detalhes de um sistema a partir dos requisitos. Como o sistema será construído.
- **Conceitos de Projeto:**
    - **Abstração:** Gerenciar complexidade escondendo detalhes.
    - **Arquitetura:** Estrutura geral do sistema, componentes principais e suas relações. (Ex: MVC, Camadas, Cliente-Servidor).
    - **Padrões de Projeto (Design Patterns):** Soluções reutilizáveis para problemas comuns de projeto.
    - **Modularidade:** Dividir o sistema em módulos coesos e com baixo acoplamento.
    - **Encapsulamento/Ocultamento de Informação:** Esconder detalhes internos de um módulo.
    - **Refinamento:** Elaborar detalhes progressivamente.
- **Níveis de Projeto:**
    - **Projeto Arquitetural:** Visão macro, estrutura global.
    - **Projeto Detalhado:** Visão micro, detalhes de cada módulo/classe.
- **Modelos UML para Projeto:**

    - **Diagrama de Classes:** Estrutura estática, classes, atributos, métodos, relacionamentos.
    - **Diagrama de Sequência:** Interações entre objetos ao longo do tempo (dinâmico).
    - **Diagrama de Atividades:** Fluxo de trabalho ou operações (dinâmico).
    - **Diagrama de Componentes:** Organização física dos componentes.
    - **Diagrama de Implantação (Deployment):** Hardware e distribuição do software.

Aula 08: Princípios de Teste de Software

- **O que é Teste?** Processo de executar um sistema com o objetivo de encontrar defeitos (erros, falhas). Visa verificar e validar o software.
- **Objetivos do Teste:** Encontrar defeitos, fornecer confiança na qualidade, prevenir defeitos, verificar se requisitos foram atendidos.
- **Princípios Gerais do Teste:**
    - Teste mostra a presença de defeitos, não a ausência.
    - Teste exaustivo (todas as combinações) é impossível.
    - Testar cedo economiza tempo e dinheiro.
    - Defeitos tendem a se agrupar (Princípio de Pareto).
    - Paradoxo do Pesticida: Testes repetidos perdem eficácia.
    - Teste depende do contexto.
    - Falácia da ausência de erros (software sem erros pode ser inútil).
- **Níveis de Teste:**
    - **Teste de Unidade:** Testa componentes individuais (classes, métodos). Feito por desenvolvedores.
    - **Teste de Integração:** Testa a interação entre componentes/módulos.
    - **Teste de Sistema:** Testa o sistema completo em relação aos requisitos funcionais e não funcionais.
    - **Teste de Aceitação:** Validação pelo cliente ou usuário final para verificar se o sistema atende às suas necessidades.

Aula 09: Técnicas de Teste de Software

- **Caso de Teste:** Conjunto de entradas, condições de execução, resultados esperados e pós-condições, desenvolvido para um objetivo específico.
- **Estratégias/Categorias de Técnicas:**
    - **Caixa-Preta (Black-Box):** Baseada na especificação (requisitos), sem conhecimento do código interno. Testa a funcionalidade.
        - **Particionamento de Equivalência:** Divide dados de entrada em classes equivalentes. Testa um valor de cada classe.
        - **Análise de Valor Limite (Boundary Value Analysis):** Testa os valores nos limites das partições (e adjacentes).
        - **Tabela de Decisão:** Testa combinações de condições de entrada e ações resultantes.

        - **Teste de Transição de Estado:** Testa mudanças de estado do sistema.
    - **Caixa-Branca (White-Box):** Baseada na estrutura interna do código (lógica, caminhos). Requer acesso ao código.
        - **Cobertura de Comando (Statement Coverage):** Executar cada linha de código pelo menos uma vez.
        - **Cobertura de Decisão/Desvio (Branch Coverage):** Executar cada decisão (IF, loop) para verdadeiro e falso.
        - **Cobertura de Caminho (Path Coverage):** Executar cada caminho independente possível no código. (Geralmente inviável).
    - **Baseada na Experiência:** Usa conhecimento e intuição do testador.
        - **Teste Exploratório:** Aprendizado, projeto e execução de testes simultâneos.
        - **Adivinhação de Erro (Error Guessing):** Tentar prever erros comuns.

Resumo Engenharia de Software (Aulas 01-09) - Baseado em Herysson/Engenharia-de-Software

---

Aula 01: Introdução à Engenharia de Software (ES)

- **O que é ES?** Disciplina da engenharia focada em todos os aspectos da produção de software, desde a concepção até a manutenção, buscando qualidade, custo e prazo adequados.
- **Software:** Programas + Documentação + Dados de configuração.
- **Crise do Software:** Dificuldades históricas (custo, prazo, qualidade) no desenvolvimento de sistemas complexos. ES surge como resposta.
- **Diferença ES vs Ciência da Computação:** ES = Foco na prática e construção de sistemas; CC = Foco nos fundamentos teóricos.
- **Qualidade de Software:** Atributos como funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade, portabilidade.
- **Ética e Profissionalismo:** Importância do comportamento ético na prática da ES.

Aula 02: Ciclo de Vida e Modelos de Processo de Software

- **Processo de Software:** Conjunto de atividades e resultados associados para produzir um produto de software.
- **Ciclo de Vida:** Fases fundamentais: Especificação -> Projeto (Design) -> Implementação -> Validação (Testes) -> Evolução (Manutenção).
- **Modelos de Processo:** Estruturas para organizar as atividades do ciclo de vida.
    - **Cascata (Waterfall):** Sequencial, fases distintas e lineares. Pouca flexibilidade.
    - **Prototipagem:** Criação de protótipos para validar requisitos com o cliente. Iterativo.
    - **Espiral (Spiral):** Iterativo e baseado em riscos. Combina prototipagem e cascata. Complexo.

    - **Incremental:** Entrega o software em partes funcionais (incrementos).
    - **RUP (Rational Unified Process):** Iterativo e incremental, focado em casos de uso e arquitetura. Disciplinado.
    - **(Modelos Ágeis - introdução):** Abordagens iterativas e adaptativas (serão detalhadas depois).

Aula 03: Princípios e Métodos de Desenvolvimento

- **Princípios Fundamentais da ES:**
    - **Rigor e Formalidade:** Abordagem sistemática.
    - **Separação de Interesses (Concerns):** Dividir problemas complexos em partes menores.
    - **Modularidade:** Componentes independentes e coesos.
    - **Abstração:** Focar no essencial, esconder detalhes.
    - **Antecipação a Mudanças:** Projetar para facilitar evoluções.
    - **Generalidade:** Criar soluções reutilizáveis.
    - **Incrementalidade:** Desenvolver em partes.
- **Métodos de Desenvolvimento:**
    - **Estruturado:** Foco em funções e fluxo de dados (DFDs). Abordagem mais antiga.
    - **Orientado a Objetos (OO):** Foco em classes e objetos, encapsulamento, herança, polimorfismo. Usa UML.
    - **Ágil:** Foco em colaboração, resposta rápida a mudanças, entregas frequentes (Ex: Scrum, XP).

Aula 04: Princípios de Modelagem de Software

- **O que é Modelagem?** Criar representações simplificadas (modelos) de um sistema ou de parte dele.
- **Por que Modelar?** Para entender, comunicar, analisar e documentar o sistema. Ajuda a lidar com a complexidade.
- **Princípios da Modelagem:**
    - Escolher modelos adequados ao problema.
    - Diferentes modelos revelam diferentes aspectos.
    - Nível de precisão depende do objetivo.
    - Modelos devem estar conectados à realidade.
- **UML (Unified Modeling Language):** Linguagem padrão para visualização, especificação, construção e documentação de artefatos de software. Não é um método, mas uma notação.
- **Tipos de Modelos:**
    - **Estruturais:** Mostram a estrutura estática (Ex: Diagrama de Classes).
    - **Comportamentais:** Mostram a dinâmica do sistema (Ex: Diagrama de Casos de Uso, Sequência, Atividade).

Aula 05: Requisitos de Software

- **O que são Requisitos?** Descrições do que o sistema deve fazer (serviços) e das restrições sob as quais ele deve operar.
- **Tipos de Requisitos:**
    - **Funcionais:** Declaram funcionalidades ou serviços. O *que* o sistema faz. (Ex: "O sistema deve permitir cadastrar cliente").
    - **Não Funcionais (RNF):** Restrições sobre os serviços ou qualidades do sistema. O *como* o sistema faz. (Ex: Performance, segurança, usabilidade, confiabilidade).
    - **De Domínio:** Vêm do domínio da aplicação, podem ser funcionais ou não. (Ex: Uma lei específica da área).
- **Engenharia de Requisitos (ER):** Processo de descobrir, analisar, documentar, validar e gerenciar requisitos.
    - **Fases:** Elicitação -> Análise -> Especificação -> Validação -> Gerenciamento.
- **Características de Bons Requisitos:** Completos, Consistentes, Corretos, Realizáveis, Verificáveis, Rastreáveis, Não ambíguos, Modificáveis.

Aula 06: Casos de Uso (Use Cases)

- **O que são Casos de Uso?** Técnica para capturar requisitos funcionais descrevendo interações entre atores (usuários ou sistemas externos) e o sistema para alcançar um objetivo específico.
- **Elementos Principais:**
    - **Ator:** Quem ou o que interage com o sistema.
    - **Caso de Uso:** Uma funcionalidade/objetivo específico (verbo + objeto, ex: "Cadastrar Cliente").
    - **Sistema:** Fronteira que delimita o software.
    - **Relacionamentos:** Include (inclusão obrigatória), Extend (extensão opcional), Generalização (herança entre atores ou casos de uso).
- **Diagrama de Casos de Uso (UML):** Representação gráfica das interações entre atores e casos de uso. Visão geral.
- **Especificação de Caso de Uso (Textual):** Descrição detalhada de um caso de uso: Nome, Atores, Pré-condições, Pós-condições, Fluxo Principal (caminho feliz), Fluxos Alternativos (exceções, variações).

Aula 07: Modelagem e Projeto de Software (Design)

- **O que é Projeto (Design)?** Processo de definir a arquitetura, componentes, interfaces e outros detalhes de um sistema a partir dos requisitos. Como o sistema será construído.
- **Conceitos de Projeto:**
    - **Abstração:** Gerenciar complexidade escondendo detalhes.
    - **Arquitetura:** Estrutura geral do sistema, componentes principais e suas relações. (Ex: MVC, Camadas, Cliente-Servidor).

    - **Padrões de Projeto (Design Patterns):** Soluções reutilizáveis para problemas comuns de projeto.
    - **Modularidade:** Dividir o sistema em módulos coesos e com baixo acoplamento.
    - **Encapsulamento/Ocultamento de Informação:** Esconder detalhes internos de um módulo.
    - **Refinamento:** Elaborar detalhes progressivamente.
- **Níveis de Projeto:**
    - **Projeto Arquitetural:** Visão macro, estrutura global.
    - **Projeto Detalhado:** Visão micro, detalhes de cada módulo/classe.
- **Modelos UML para Projeto:**
    - **Diagrama de Classes:** Estrutura estática, classes, atributos, métodos, relacionamentos.
    - **Diagrama de Sequência:** Interações entre objetos ao longo do tempo (dinâmico).
    - **Diagrama de Atividades:** Fluxo de trabalho ou operações (dinâmico).
    - **Diagrama de Componentes:** Organização física dos componentes.
    - **Diagrama de Implantação (Deployment):** Hardware e distribuição do software.

Aula 08: Princípios de Teste de Software

- **O que é Teste?** Processo de executar um sistema com o objetivo de encontrar defeitos (erros, falhas). Visa verificar e validar o software.
- **Objetivos do Teste:** Encontrar defeitos, fornecer confiança na qualidade, prevenir defeitos, verificar se requisitos foram atendidos.
- **Princípios Gerais do Teste:**
    - Teste mostra a presença de defeitos, não a ausência.
    - Teste exaustivo (todas as combinações) é impossível.
    - Testar cedo economiza tempo e dinheiro.
    - Defeitos tendem a se agrupar (Princípio de Pareto).
    - Paradoxo do Pesticida: Testes repetidos perdem eficácia.
    - Teste depende do contexto.
    - Falácia da ausência de erros (software sem erros pode ser inútil).
- **Níveis de Teste:**
    - **Teste de Unidade:** Testa componentes individuais (classes, métodos). Feito por desenvolvedores.
    - **Teste de Integração:** Testa a interação entre componentes/módulos.
    - **Teste de Sistema:** Testa o sistema completo em relação aos requisitos funcionais e não funcionais.
    - **Teste de Aceitação:** Validação pelo cliente ou usuário final para verificar se o sistema atende às suas necessidades.

Aula 09: Técnicas de Teste de Software

- **Caso de Teste:** Conjunto de entradas, condições de execução, resultados esperados e pós-condições, desenvolvido para um objetivo específico.
- **Estratégias/Categorias de Técnicas:**
    - **Caixa-Preta (Black-Box):** Baseada na especificação (requisitos), sem conhecimento do código interno. Testa a funcionalidade.
        - **Particionamento de Equivalência:** Divide dados de entrada em classes equivalentes. Testa um valor de cada classe.
        - **Análise de Valor Limite (Boundary Value Analysis):** Testa os valores nos limites das partições (e adjacentes).
        - **Tabela de Decisão:** Testa combinações de condições de entrada e ações resultantes.
        - **Teste de Transição de Estado:** Testa mudanças de estado do sistema.
    - **Caixa-Branca (White-Box):** Baseada na estrutura interna do código (lógica, caminhos). Requer acesso ao código.
        - **Cobertura de Comando (Statement Coverage):** Executar cada linha de código pelo menos uma vez.
        - **Cobertura de Decisão/Desvio (Branch Coverage):** Executar cada decisão (IF, loop) para verdadeiro e falso.
        - **Cobertura de Caminho (Path Coverage):** Executar cada caminho independente possível no código. (Geralmente inviável).
    - **Baseada na Experiência:** Usa conhecimento e intuição do testador.
        - **Teste Exploratório:** Aprendizado, projeto e execução de testes simultâneos.
        - **Adivinhação de Erro (Error Guessing):** Tentar prever erros comuns.