Visão geral sobre a engenharia de software

Definição: Engenharia de Software trata da aplicação de abordagens sistemáticas, disciplinadas e quantificáveis para desenvolver, operar, manter e evoluir software. Ou seja, Engenharia de Software é a área da Computação que se preocupa em propor e aplicar princípios de engenharia na construção de software de forma produtiva e com qualidade.

As dificuldades essenciais (São os desafios permanentes da construção de software) são as seguintes:
Tornam Engenharia de Software diferente de outras engenharias.

- **Complexidade**: Software é uma das construções humanas (mais desafiadoras e complexas), superando até certas engenharias tradicionais.
- **Conformidade**: Software precisa se adaptar constantemente a mudanças no ambiente, como novas leis ou regulamentações. (adaptar ao seu ambiente, que muda a todo momento no mundo moderno)
- **Facilidade de mudanças:** Sistemas de software devem evoluir continuamente, incorporando novas funcionalidades conforme crescem em sucesso. (necessidade de evoluir sempre)
- **Invisibilidade**: Por ser abstrato, é difícil (visualizar e estimar o esforço necessário para construir um sistema de software.)

As dificuldades acidentais (São problemas temporários e solucionáveis) do desenvolvimento de software estão relacionadas a problemas tecnológicos, como ferramentas mal documentadas, bugs em IDEs ou interfaces pouco intuitivas, e podem ser resolvidas por engenheiros de software treinados com acesso adequado a tecnologias e recursos.

O que se estuda em Engenharia de Software?

- Engenharia de Requisitos. Projeto de Software. Construção de Software. Testes de Software. Manutenção de Software. Gerência de Configuração. Gerência de Projetos. Processos de Software. Modelos de Software. Qualidade de Software. Prática Profissional. Aspectos Econômicos.

Engenharia de Requisitos

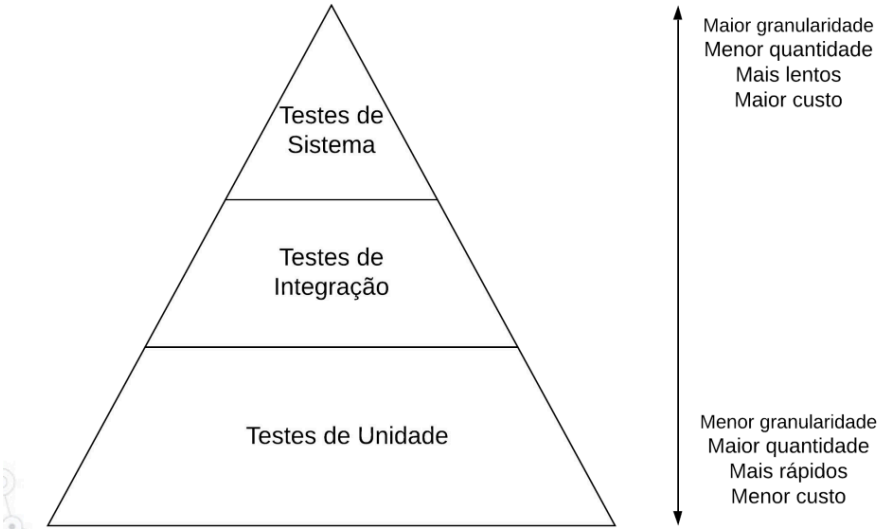
Os requisitos de um sistema definem o que ele deve fazer e como ele deve operar.

Requisitos funcionais definem o que um sistema deve fazer; isto é, quais funcionalidades ou serviços ele deve implementar.

Já os requisitos não-funcionais definem como um sistema deve operar, sob quais restrições e com qual qualidade de serviço. São exemplos de requisitos não-funcionais: desempenho, disponibilidade, tolerância a falhas, segurança, privacidade, interoperabilidade, capacidade, manutenibilidade e usabilidade.

Testes de Software

Consiste em executar um programa com casos específicos para verificar se ele se comporta como esperado, lembrando que testes mostram a presença de bugs, mas não a sua ausência.



Manutenção de Software

- ◎ **Corretiva**: Tipo de manutenção focada em corrigir bugs e defeitos encontrados após o software ser entregue.
- ◎ **Adaptativa**: Modificar o software para que ele se adapte a mudanças no seu ambiente externo.
- ◎ **Evolutiva**: Adicionar novas funcionalidades ou melhorar as existentes para atender a novas necessidades dos utilizadores.
- ◎ **Refactoring (Preventiva)**: Melhorar a estrutura interna do código sem alterar o seu comportamento externo, para facilitar a manutenção futura e reduzir a 'dívida técnica'.

Processos de Desenvolvimento de Software

- **Processo de software:** define as atividades que devem ser seguidas para construir um sistema de software.

Construção de Software

- Construção trata da implementação, isto é, codificação do sistema. Neste momento, existem diversas decisões que precisam ser tomadas, como, por exemplo: definir os algoritmos e estruturas de dados que serão usados, definir os frameworks e bibliotecas de terceiros que serão usados.

Tipos de Processos

- **Waterfall (Cascata)**: método sequencial inspirado em engenharias tradicionais, com etapas como levantamento de requisitos, análise, projeto detalhado, codificação e testes, liberando o sistema somente ao final. Apresenta problemas de adaptação a mudanças, já que a validação ocorre tardiamente.
- **Ágeis (Iterativos e Incrementais):** Construindo o software em pequenos incrementos, revisados e validados pelos usuários periodicamente.
- **Enquanto Waterfall segue um planejamento rígido e sequencial, os processos ágeis focam em ciclos curtos de entrega e feedback contínuo, aumentando a flexibilidade e qualidade do software.**

Sistemas C (Casuais)

São pequenos, de baixo risco e sem pressão por alta qualidade.

Podem ter bugs, código duplicado ou interfaces simples, geralmente desenvolvidos por 1-2 programadores.

Exemplos: um script para trabalho acadêmico, um programa de conversão de arquivos usado uma única vez, ou um sistema simples para gerenciar salas de reunião na empresa.

Não exigem técnicas avançadas de Engenharia de Software; o maior risco é over-engineering, ou seja, usar soluções complexas onde não são necessárias.

Sistemas A (Acute / Missão crítica)

Falhas podem causar grandes prejuízos ou até perda de vidas.

Exigem processos rígidos, revisões de código, certificação externa, redundância de hardware/software e, muitas vezes, especificações formais.

Exemplos: sistemas de controle de carros autônomos, usinas nucleares, aviões, equipamentos de UTI ou trens de metrô.

Sistemas B (Business)

São sistemas corporativos, aplicativos web, bibliotecas ou softwares de uso geral, sendo os mais comuns em empresas.

Podem se beneficiar das boas práticas de Engenharia de Software, aumentando produtividade, qualidade interna (manutenção e legibilidade) e externa (menos bugs em produção).

Exemplos: sistemas de gestão financeira, recursos humanos, logística, sistemas de e-commerce, redes sociais, editores de texto ou planilhas, IDEs e bancos de dados.

Modelos de processo de software

Um Modelo de Processo de Software é uma representação simplificada e estruturada das etapas, atividades, e tarefas envolvidas na produção de um software, desde a sua concepção inicial até a entrega final e manutenção.

1. Modelo em Cascata (Waterfall)

Sequencial e linear para o desenvolvimento de software. Nele, as fases do projeto são executadas em uma ordem estrita.

Ordem fixa, onde cada fase deve ser totalmente concluída antes que a próxima seja iniciada. Esse fluxo unidirecional.

Etapas e Fluxo do Modelo

0.1 Levantamento de Requisitos
0.2 Design (Projeto)
0.3 Implementação (Desenvolvimento)
0.4 Testes
0.5 Manutenção

Vantagens

- Ênfase na documentação e Estrutura clara e organizada

Desvantagens

- Rigidez e pouca flexibilidade e Demora na entrega e Riscos acumulados.

Cenário Adequado para Aplicação: Sistemas militares: lançamento de mísseis, radares de defesa

Maior Risco de Usá-lo

- O maior risco reside na possibilidade de os requisitos do cliente não terem sido completamente compreendidos no início ou mudarem ao longo do tempo, levando a um produto final que não atende às suas reais necessidades.

2. Modelo V

Extensão do Modelo em Cascata, que **enfatiza a relação entre cada fase de desenvolvimento e sua fase de teste correspondente.**

A forma de "V" ilustra a validação e a verificação ocorrendo em paralelo com o desenvolvimento.

Etapas e Fluxo do Modelo

Lado Esquerdo (Desenvolvimento):

- Análise de Requisitos de Negócio
- Projeto do Sistema
- Projeto da Arquitetura
- Projeto Detalhado do Módulo
- Vértice do "V" – Codificação

Lado Direito (Testes):

- Teste de Unidade
- Teste de Integração
- Teste de Sistema
- Teste de Aceitação

Vantagens

- Requisitos definidos no início. Antecipa o planejamento de testes. Identificação precisa de erros

Desvantagens

- Baixa flexibilidade. Demora na entrega e documentação extensa

Cenário Real

Muito utilizado em sistemas críticos, onde a correspondência entre requisitos e testes garante confiabilidade e segurança. Exemplos: freios ABS e airbags, em que cada requisito de segurança precisa de um teste correspondente.

Maior Risco

Altíssimo custo de mudança, podendo atrasar ou inviabilizar o projeto.

3. Modelo Incremental

Melhoria do modelo cascata. O projeto do software é dividido em pequenas partes, com cada uma sendo entregue separadamente.

É uma metodologia baseada em incrementos, onde cada entrega adiciona novas funcionalidades ao sistema.

Etapas (cada incremento funciona como um mini-projeto com 5 fases)

- Comunicação
- Planejamento
- Modelagem
- Construção
- Emprego – Entrega ao cliente e coleta de feedback

Fluxo

- Funciona em sequências lineares escalonadas
- Cada incremento é entregue ao cliente
- O cliente avalia, fornece feedback e isso é usado para planejar o próximo incremento
- O processo se repete até que o produto esteja completo
- Resultado: refinamentos sucessivos e melhorias a cada iteração

Vantagens

- O cliente pode receber parte do projeto e já utilizá-lo
- Feedback facilitado pelo cliente, que pode avaliar e testar

Desvantagens

- A estrutura do sistema pode degradar com novos incrementos
- Tempo e dinheiro necessários para refatorações

Cenário Real

Exemplo: Desenvolvimento de aplicativos móveis

- Primeira versão: funções básicas (cadastro e visualização de restaurantes)
- Segundo incremento: adição de pagamento online
- Próximos incrementos: rastreamento de pedidos em tempo real

O cliente já utiliza o app desde o início, e cada entrega o torna mais completo, aproveitando o feedback dos usuários.

Maior Risco

- Projetos de grande porte
- Pressão por prazos e entregas rápidas

4. Modelo RAD

Modelo de processo de software incremental que enfatiza ciclos de desenvolvimento muito curtos.

O objetivo é entregar rapidamente sistemas funcionais de alta qualidade, com forte envolvimento do usuário e uso intensivo de protótipos funcionais.

Etapas do RAD

- Modelagem do Negócio Modelagem de Dados Modelagem de Processos

- Geração da Aplicação – Prototipagem rápida, ciclos de feedback e construção iterativa
  Teste e Modificação – Testes finais,

Fluxo do RAD

- Iterativo e cíclico
- Começa com uma fase rápida de modelagem
- Entra em ciclos repetitivos de prototipação:

Vantagens

- Desenvolvimento rápido: tempo entre requisitos e entrega funcional é reduzido
- Envolvimento do cliente: feedback contínuo garante aderência às expectativas

Desvantagens

- Necessita equipes experientes para lidar com ciclos curtos e prototipação
- Pouco adequado para projetos grandes e complexos.

Cenário Real

Um sistema de agendamento online foi desenvolvido em menos de 3 meses com RAD, substituindo planilhas e papel, permitindo uso rápido, entregas por partes e ajustes contínuos com feedback dos usuários.

Maior Risco

- Falta de planejamento detalhado e arquitetura sólida
- Escalabilidade limitada, prejudicando grandes projetos

5. Processo Unificado - Foca na arquitetura. Combina iterativo e incremental

Modelo de desenvolvimento de software que combina práticas iterativas, incrementais e orientadas a objetos.

Funciona como um mapa flexível para construir software complexo, sendo construído gradualmente, com cada iteração entregando uma versão funcional e mais completa do sistema.

Ciclos Iterativos e Incrementais

- O projeto é dividido em iterações (mini-projetos)
- Cada iteração entrega um sistema executável, ainda que incompleto
- O sistema evolui incrementalmente, adicionando e refinando funcionalidades a cada ciclo
- Feedback de cada iteração é usado para ajustar requisitos e arquitetura

Fases do Processo Unificado

- Inception
- Elaboração
- Construção
- Transição
- Entrega final e implantação para o usuário

Vantagens

- Permite mudanças ao longo do projeto
- Reduz riscos com entregas frequentes e revisões constante

Desvantagens

- Pode ser complexo de gerenciar sem experiência

- Exige disciplina e organização da equipe

Cenário Real

- Muito utilizado em projetos de médio a grande porte, especialmente quando há incertezas nos requisitos ou necessidade de evolução constante do sistema.

Maior Risco

- Se não for adaptado ao projeto, pode gerar burocracia excessiva ou perda de controle. Equipes inexperientes podem se perder na gestão das iterações e fases.

6. (Evolutivos) Modelo Espiral

Processo evolutivo que combina elementos dos modelos sequenciais e incrementais, com forte foco na análise e gestão de riscos. O desenvolvimento ocorre em ciclos (voltas da espiral), cada um entregando uma versão mais completa do sistema.

Etapas de Cada Ciclo

- Determinação de Objetivos
- Avaliação e Redução de Riscos
- Desenvolvimento e Validação
- Planejamento da Próxima Iteração

Vantagens

- Adequado para projetos grandes e inovadores
- Íntegra prototipação e feedback contínuo

Desvantagens

- Complexo de gerenciar
- Pode demandar mais tempo e recursos

Fluxo

- O projeto avança em espiral, repetindo as quatro etapas a cada ciclo. O processo termina quando o sistema está maduro para entrega.

Cenário Real

Muito utilizado em projetos grandes e de alto risco, como sistemas bancários, softwares embarcados em aviões ou P&D.

Maior Risco

Se a análise de riscos for mal conduzida, o projeto pode perder o controle, consumir recursos excessivos ou entrar em ciclos intermináveis.

7. (Evolutivos) Prototipação

Utiliza versões simplificadas e funcionais do sistema para validar requisitos, explorar soluções e obter feedback do usuário antes da implementação final. Reduz incertezas, alinha expectativas e refina funcionalidades por meio de iterações rápidas.

Etapas e Fluxo

- Comunicação
- Projeto Rápido
- Construção do Protótipo
- Emprego e Feedback
- Ciclo Iterativo

Vantagens

- Permite validação precoce de funcionalidades Envolvimento direto do usuário

Desvantagens

- Pode gerar expectativas irreais se o protótipo for confundido com o produto final
- Protótipos descartáveis podem aumentar custos

Cenário Real

- Usado em projetos com requisitos incertos ou difíceis de expressar, como sistemas inovadores ou com forte interação com o usuário.

Maior Risco

- O cliente pode acreditar que o protótipo já é o produto final, causando frustração. Além disso, se mal gerenciado, pode gerar retrabalho e aumento de custos.

Metodologias Ágeis

Engenharia Tradicional é o modelo clássico, sequencial e rígido, em que tudo é planejado e documentado antes da implementação, com pouca flexibilidade para mudanças.
Engenharia de Software ≠ Engenharia Tradicional

2. O Manifesto Ágil

- A ideia central do Ágil é o **desenvolvimento iterativo e incremental**, onde o sistema é construído em pequenas partes funcionais que são validadas continuamente com o usuário
- **Indivíduos e interações** mais que processos e ferramentas.
- **Software em funcionamento** mais que documentação abrangente.
- **Colaboração com o cliente** mais que negociação de contratos.
- **Responder a mudanças** mais que seguir um plano.

3. Métodos Ágeis

- São processos que aplicam os princípios do Manifesto Ágil, definindo papéis, eventos e práticas específicas. Os principais são XP, Scrum e Kanban.
- **Extreme Programming (XP) = Valores + Princípios + Práticas**
- Metodologia ágil focada em **alta qualidade de código e entregas rápidas**, com práticas como programação em par, testes automáticos e integração contínua
- Método que leva práticas de desenvolvimento ao "extremo". Ele é baseado em:

- **Valores Fundamentais:** Comunicação, Simplicidade, Feedback, Coragem e Respeito e qualidade de vida.
- **Princípios:** Melhorias contínuas, passos pequenos (*baby steps*), e responsabilidade pessoal.
- **Práticas:**
    - **Programação Pareada (Pair Programming):** Dois desenvolvedores trabalham juntos em um único computador. Um é o "piloto" (escreve o código) e o outro é o "navegador" (revisa e orienta). Isso ajuda a reduzir bugs e disseminar conhecimento.
    - **Desenvolvimento Orientado a Testes (TDD):** Os testes automatizados são escritos *antes* do código da funcionalidade.

- **Integração Contínua (CI):** O código desenvolvido é integrado ao projeto principal de forma frequente.

- **Contratos de Escopo Aberto:** Diferente do escopo fechado (preço e prazo fixos), XP defende contratos onde o escopo é definido a cada iteração e o pagamento é por tempo (homem/hora), privilegiando a qualidade e a flexibilidade.

Práticas sobre o Processo de Desenvolvimento	Práticas de Programação	Práticas de Gerenciamento de Projetos

Contratos de software podem ser internos ou terceirizados, e se dividem em:

- **Escopo Fechado:** o cliente define os requisitos no início, e eles permanecem fixos.
- **Escopo Aberto:** os requisitos podem evoluir durante o projeto, com colaboração entre cliente e equipe (Escopo definido a cada iteração).

Scrum

Framework ágil para gestão de projetos, baseado em iterações curtas chamadas sprints, com foco em entregas incrementais, colaboração da equipe e adaptação contínua às mudanças.

Principal evento: Sprints - Um ciclo de desenvolvimento com duração fixa onde um incremento funcional do produto é criado.

O que se faz em um sprint?

● Implementa-se algumas histórias dos usuários
● Histórias = funcionalidades do sistema
● Exemplo: fórum de perguntas e resposta

Papéis do Time Scrum:

(PO): O especialista no negócio. É responsável por criar e priorizar os itens do backlog (funcionalidades), garantindo que o time construa o produto certo.

- Durante sprint, PO explica histórias para devs. Conversas entre PO e devs

● Escrever histórias dos usuários ● Definir "testes de aceitação" de histórias

Scrum Master: Um "líder servidor" especialista em Scrum. Ajuda o time a seguir o processo, remove impedimentos e não é o chefe da equipe.

Desenvolvedores (Devs): Os profissionais que constroem o produto (programadores, designers, etc.). São auto-organizáveis e multidisciplinares.

Backlog do Produto: Lista de histórias do usuário

● (e outros itens de trabalho importantes)
● Duas características:
○ Priorizada: histórias do topo têm maior prioridade
○ Dinâmica: histórias podem sair e entrar…

● Em um time Scrum, todos têm o mesmo nível hierárquico; PO não é o chefe dos Devs; Devs têm autonomia para dizer que não vão conseguir implementar tudo que o PO quer em um único sprint.

- **Artefatos do Scrum:**
- Product Backlog: Uma lista dinâmica e priorizada de tudo o que é desejado no produto, escrita na forma de Histórias de Usuário. As histórias no topo têm maior prioridade.
- Sprint Backlog: A lista de tarefas que os desenvolvedores se comprometem a realizar durante um sprint para implementar as histórias selecionadas.
- **Sprint termina com dois eventos:**

Planejamento do Sprint (Sprint Planning)
Reunião Diária (Daily Scrum)
1.Review: mostrar o sprint → histórias aprovadas ou voltam ao backlog.
2.Retrospectiva: discutir o que deu certo e melhorar → foco em evolução.
Conceitos Essenciais

- **Time-box:** eventos com duração fixa.
- **Scrum Board:** quadro visual para acompanhar progresso das histórias.
- **Story Points:** estimam o tamanho das histórias e ajudam a planejar o sprint; definidos empiricamente pelo time.

Kanban = "cartão visual" → quadro de post its
Método de gestão visual de tarefas, que utiliza quadros e cartões (post-its) para acompanhar o progresso e controlar o fluxo de trabalho de forma clara e contínua.
Diferenças para o Scrum: Não possui sprints, papéis ou eventos obrigatórios. É um método evolucionário que começa com o fluxo de trabalho atual da equipe.

Sua ideia central é o **sistema "pull"**, onde os membros da equipe "puxam" as tarefas para si assim que ficam disponíveis, em vez de o trabalho ser "empurrado" para eles.

- **Quadro Kanban:** É a principal ferramenta, um quadro visual com colunas que representam cada etapa do processo (Ex: Backlog, Implementação, Revisão). As tarefas se movem da esquerda para a direita.
- **Limites WIP (Work in Progress):** Esta é a regra mais importante. Cada etapa do fluxo tem um número máximo de tarefas permitidas. Isso evita sobrecarga, expõe gargalos e força a equipe a focar em concluir o trabalho em andamento, seguindo a frase: **"pare de começar e comece a terminar"**.
- **Flexibilidade:** É um método mais simples e flexível que o Scrum, sem papéis ou eventos obrigatórios, sendo mais adequado para equipes maduras.