PRINCÍPIOS BÁSICOS DO DESENVOLVIMENTO PARA PROJETOS LOW-CODE

Prof. Bruno Villanova

1

AGENDA

- Requisitos
- Design
- Codificação
- Testes
- Controle de versão
- Manutenção e Evolução
- Documentação
- Gerenciamento de projetos
- Segurança
- Implantação e entrega contínua

2

3

REQUISITOS

Compreensão e documentação clara dos objetivos e funcionalidades que o software deve atender.

Divididos em Requisitos Funcionais e Requisitos não funcionais.

Requisitos Funcionais vs. Não Funcionais: Diferenciação entre as funcionalidades desejadas e as características não relacionadas diretamente às funcionalidades, como desempenho e segurança.

4

REQUISITOS FUNCIONAIS

- Descrevem as funcionalidades específicas que um sistema, software ou produto deve oferecer.

"O que eu preciso fazer para que o meu sistema atenda as necessidades do meu usuário ou cliente?"

Exemplo: para um sistema de gerenciamento de tarefas, um requisito funcional seria o CRUD das tarefas.

5

REQUISITOS NÃO FUNCIONAIS

- Referem-se as características que não estão diretamente relacionadas às funcionalidades específicas do sistema, mas sim a suas qualidades ou restrições.
- Definem critérios de desempenho, usabilidade, segurança entre outros.

Exemplo: Ainda no nosso sistema de gerenciamento de tarefas, um requisito não funcional seria a resposta para o cliente levar menos de 2 segundos.

6

TABELA COMPARATIVA

Aspecto	Requisitos Funcionais	Requisitos Não Funcionais
O que definem?	Funcionalidades específicas e comportamentos do sistema.	Características que não estão diretamente relacionadas às funcionalidades, mas sim a suas qualidades gerais.
Exemplos	Criar, editar, excluir tarefas (sistema de gerenciamento).	Tempo de resposta, segurança, escalabilidade, usabilidade.
Medidos por	O que o sistema faz.	Como o sistema realiza suas funções.
Relacionamento com o usuário	Diretamente ligados às necessidades e objetivos do usuário.	Impactam a experiência do usuário, mas de maneira indireta.
Alterações frequentes?	Sujeitos a mudanças frequentes conforme evolução do sistema.	Geralmente mais estáveis, alterações menos frequentes.
Impacto nas Funcionalidades	Alterações podem afetar diretamente o que o sistema faz.	Alterações têm impacto nas características e qualidades gerais do sistema.

7

DESIGN

8

DESIGN

Dividido em:

- Arquitetura de Software
- Design de interface (UI)
- Experiência do usuário (UX)

9

ARQUITETURA DE SOFTWARE

A arquitetura de software refere-se à estrutura geral e organização do sistema.

Inclui a identificação e a inter-relação dos principais componentes, módulos e seus padrões de interação.

Uma arquitetura bem projetada define a base para o desenvolvimento do software, fornecendo uma visão clara de como as partes do sistema se encaixam.

- Escalabilidade
- Manutenção facilitada
- Desempenho otimizado

10

UI – DESIGN DE INTERFACE E UX – EXPERIÊNCIA DO USUÁRIO

O design de UI refere-se à aparência visual do software, enquanto o design de UX abrange a experiência geral do usuário durante a interação com o sistema. Ambos são cruciais para garantir que o software seja intuitivo, acessível e agradável de usar.

- Usabilidade: eficiência, baixa taxa de erros e acessibilidade.
- Satisfação do usuário: fidelidade, engajamento.
- Retenção de usuários: crescimento sustentável, valor contínuo, maior envolvimento, menor taxa de desistência.

11

CODIFICAÇÃO

12

CODIFICAÇÃO

As plataformas de desenvolvimento low-code revolucionaram a maneira como o software é construído, reduzindo significativamente a necessidade de codificação extensa.

Importante:

- **Escolha da linguagem de programação:** para implementar os requisitos do software sob customização dentro das plataformas de low-code.
- **Boas Práticas de Codificação:** Adoção de padrões e convenções para garantir código legível, eficiente e fácil de manter.

- Consistência
- Modularidade: divida seu código em módulos.
- Tratamento de erros
- Testes
- Documentação
- Controle de versão
- Segurança
- Melhoria contínua

13

CARACTERÍSTICAS LOW-CODE

- **Desenvolvimento visual:** Interfaces de arrastar e soltar permitem que os usuários projetem aplicações visualmente.
- **Componentes pré-construídos:** As plataformas low-code vêm com uma biblioteca de componentes e templates pré-construídos, como elementos de UI, conectores de dados e lógica de negócios. Isso acelera o desenvolvimento ao reutilizar elementos prontos.
- **Automação e Integração**: Ferramentas de automação e integração simplificam a conexão com sistemas externos e a automação de processos de negócios, reduzindo ainda mais a necessidade de codificação manual.
- **Extensibilidade**: Embora o low-code minimize a codificação, ele ainda permite extensibilidade através de scripts personalizados e integração de APIs.

14

TESTES

15

TESTES DE SOFTWARE

Os testes de software são uma etapa essencial no processo de desenvolvimento de software, onde o código é avaliado para garantir que atenda aos requisitos de qualidade e funcionalidade esperados.

Tipos de testes:

- Unitários.
- Integrados.
- Aceitação.
- Regressão.
- Desempenho.
- Segurança.

16

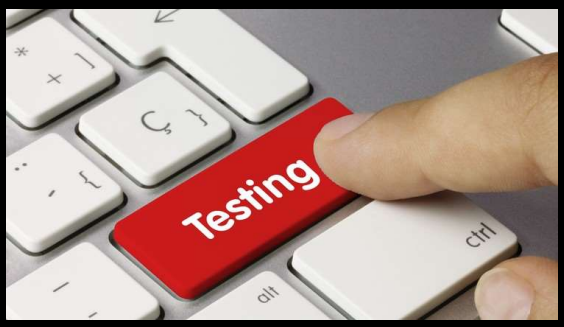
TESTES UNITÁRIOS

Os testes unitários avaliam unidades individuais de código, como funções ou métodos, para garantir que operem conforme o esperado.

Os testes unitários são frequentemente automatizados e realizados durante o desenvolvimento.

17

TESTES DE INTEGRAÇÃO



Verificam a interação entre diferentes partes do sistema, garantindo que funcionem corretamente.

Isso inclui testes de interfaces, comunicação entre módulos e integração com sistemas externos.

18

TESTES DE ACEITAÇÃO

Avaliam se o software atende aos requisitos de negócios e às expectativas do cliente.

Esses testes são conduzidos para garantir que o software entregue atenda aos critérios de aceitação definidos.

19

TESTES DE REGRESSÃO

Verificam se as alterações ou correções recentes não introduziram novos defeitos ou impactaram negativamente o funcionamento de partes do sistema que estavam funcionando corretamente anteriormente.

20

TESTES DE DESEMPENHO

Avaliam o desempenho do software em condições específicas, como carga pesada, tempo de resposta e escalabilidade.

Isso ajuda a identificar gargalos de desempenho e otimizar o software para lidar com volumes de dados ou tráfego de usuários crescentes.

21

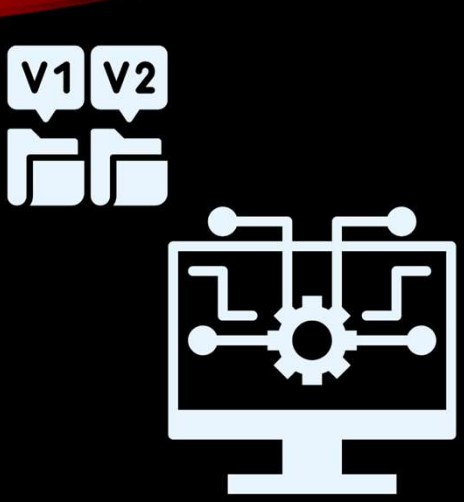
TESTES DE SEGURANÇA

Analisam a segurança do software, identificando vulnerabilidades, falhas de autenticação, possíveis brechas de segurança e protegendo o sistema contra ataques maliciosos.

22

CONTROLE DE VERSÃO

23



CONTROLE DE VERSÃO

O controle de versões de software é um sistema que permite gerenciar e acompanhar as mudanças feitas no código-fonte de um projeto ao longo do tempo.

Essas alterações são armazenadas em um repositório, onde desenvolvedores podem colaborar, acessar e controlar diferentes versões do código.

24

CONTROLE DE VERSÃO

Funcionalidades:

- Repositório.
- Commit.
- Branch.
- Merge.
- Conflitos.
- Revisão de código.
- Histórico de alterações.

25

CONTROLE DE VERSÃO

Exemplos de ferramentas para controle de versões de softwares:

- Git.
- SVN (Subversion).
- Mercurial.
- Perforce Helix Core.
- Microsoft Team Foundation Version Control.

26

CONTROLE DE VERSÃO PARA NO - CODE

Para plataformas de desenvolvimento no-code, onde o desenvolvimento de aplicativos é feito sem a necessidade de escrever código, as ferramentas de controle de versão podem funcionar de maneira um pouco diferente em comparação com o desenvolvimento tradicional.

Bubble e Adalo não possuem um controle de versões interna e muitos devs utilizam o Git para tal feito.

Notion e Airtable são utilizadas para controlar histórico de edições e restaurar versões anteriores conforme necessário.

27

MANUTENÇÃO E EVOLUÇÃO

28

MANUTENÇÃO E EVOLUÇÃO

A manutenção e evolução do software referem-se às atividades realizadas após o lançamento inicial de um produto de software, visando garantir seu funcionamento contínuo, corrigir problemas, implementar novos recursos e adaptá-lo às mudanças nos requisitos e no ambiente operacional.

29

MANUTENÇÃO E EVOLUÇÃO

Pontos importantes:

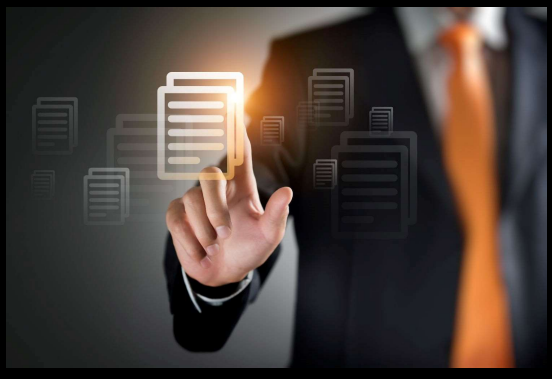
- Correção de defeitos.
- Atualizações de segurança.
- Melhorias de desempenho.
- Adição de novas funcionalidades.
- Adaptação a mudanças.
- Gerenciamento de versões.

30

DOCUMENTAÇÃO

31

DOCUMENTAÇÃO



É uma parte essencial do processo de desenvolvimento, pois fornece informações detalhadas sobre o funcionamento, arquitetura, design e outros aspectos relevantes do projeto.

Pontos importantes:

- Documentação de requisitos.
- Documentação de design.
- Documentação de código.
- Documentação de testes.
- Documentação de implantação e operação.
- Documentação do usuário.

32

DOCUMENTAÇÃO DE REQUISITOS

Inclui a descrição detalhada dos requisitos funcionais e não funcionais do sistema.

- Necessidade do usuário.
- Critérios de aceite.
- Casos de uso.
- Diagramas de fluxos de dados.
- Processos de negócio.

33

PASSO A PASSO PARA DOCUMENTAR REQUISITOS

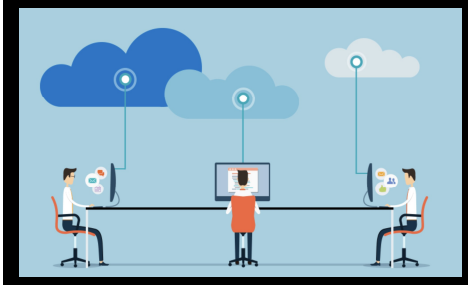
34

DOCUMENTAÇÃO DE DESIGN

A documentação de design é uma parte essencial do processo de desenvolvimento de software, pois descreve a arquitetura, estrutura e detalhes de implementação do sistema.

35

DOCUMENTAÇÃO DE DESIGN



Arquitetura do sistema: A documentação de design descreve a arquitetura geral do sistema, incluindo a organização de componentes, módulos e camadas, bem como as interações entre eles.

Detalhes de componentes e módulos: Para cada componente ou módulo do sistema, a documentação de design fornece detalhes sobre sua funcionalidade, responsabilidades, interfaces e dependências.

36

DOCUMENTAÇÃO DE DESIGN

Padrões de Design e Práticas recomendadas: A documentação de design pode incluir diretrizes, padrões de design e práticas recomendadas a serem seguidas durante o desenvolvimento do sistema.

Diagramas de Sequência e Fluxo: Além da arquitetura geral do sistema, a documentação de design pode incluir diagramas de sequência e fluxo que descrevem o comportamento e a interação entre os diferentes componentes do sistema em resposta a eventos e entradas específicas.

37

DOCUMENTAÇÃO DE DESIGN

Detalhes de Implementação: A documentação de design pode incluir detalhes técnicos sobre a implementação de componentes individuais, incluindo estruturas de dados, algoritmos, APIs e outros aspectos relacionados à codificação e implementação do sistema.

Considerações de Desempenho e Escalabilidade: A documentação de design pode abordar considerações de desempenho e escalabilidade do sistema, incluindo otimizações de código, técnicas de cache, dimensionamento de recursos e outras estratégias para garantir que o sistema possa lidar com cargas de trabalho crescentes e operar de forma eficiente em diferentes ambientes.

38

DOCUMENTAÇÃO DE CÓDIGO

A documentação de código é uma prática fundamental no desenvolvimento de software que visa fornecer informações claras e compreensíveis sobre o funcionamento, estrutura e uso do código-fonte de um sistema.

39

DOCUMENTAÇÃO DE CÓDIGO

```
namespace TestaDoc
{

    /// <summary>
    /// Classe usada para envio de emails.
    /// </summary>
     public class Emails
    {
    }
}
```

```
/// <summary>
///
/// </summary>
public class Emails
{
    /// <summary>
    ///
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="recipient"></param>
    /// <param name="subject"></param>
    /// <param name="message"></param>
    /// <returns></returns>
    public bool SendEmail(string sender, string recipient, string subject,string message)
    {
        //Código do método entra aqui
    }
}
```

Comentários Inline: Os comentários são trechos de texto inseridos diretamente no código-fonte para explicar sua funcionalidade, lógica e propósito.

Documentação de Funções e Métodos: Cada função ou método no código deve ser acompanhado de uma descrição que explique o que ela faz, quais são seus parâmetros e qual é o seu retorno.

Convenções de Nomenclatura: A documentação de código também pode incluir informações sobre as convenções de nomenclatura utilizadas no projeto, como nomes de variáveis, funções, classes e outros elementos.

40

DOCUMENTAÇÃO DE CÓDIGO

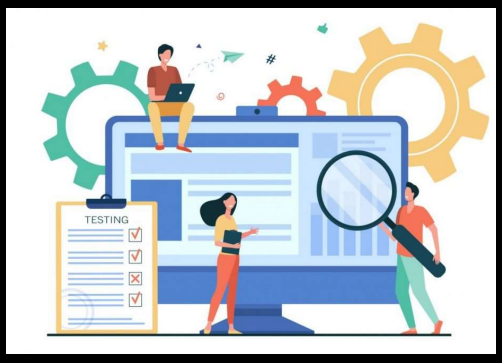
Exemplos de Uso: Quando apropriado, a documentação de código pode incluir exemplos de uso que demonstram como chamar uma função ou usar uma classe em diferentes cenários.

Diagramas e Fluxogramas: Em alguns casos, a documentação de código pode incluir diagramas e fluxogramas que ilustram a estrutura ou fluxo de dados do sistema.

Ferramentas de Geração de Documentação: Existem várias ferramentas disponíveis para ajudar na geração automática de documentação de código a partir dos comentários no código-fonte.

41

DOCUMENTAÇÃO DE TESTES



A documentação de testes é uma parte crucial do processo de desenvolvimento de software, pois fornece informações detalhadas sobre os casos de teste, procedimentos de teste e resultados de testes realizados durante o desenvolvimento do sistema.

42

DOCUMENTAÇÃO DE TESTES

Casos de Teste: A documentação de testes inclui uma lista de casos de teste detalhados que descrevem as condições de entrada, os passos a serem seguidos e os resultados esperados para cada funcionalidade ou componente do sistema.

Procedimentos de Teste: Além dos casos de teste individuais, a documentação de testes também descreve os procedimentos gerais de teste a serem seguidos durante o processo de teste.

Resultados de Teste: A documentação de testes registra os resultados de cada teste realizado, incluindo quais casos de teste foram executados, se eles passaram ou falharam e quaisquer observações ou problemas encontrados durante o teste.

43

DOCUMENTAÇÃO DE TESTES

Rastreamento de Defeitos: Quando um teste revela um defeito ou problema no sistema, a documentação de testes é atualizada para registrar detalhes sobre o defeito, incluindo sua gravidade, causa raiz, passos para reproduzir e quaisquer soluções ou correções aplicadas.

Relatórios de Teste: A documentação de testes também pode incluir relatórios de teste resumidos que fornecem uma visão geral do estado do teste, incluindo o número total de casos de teste, quantos foram executados com sucesso, quantos falharam e quaisquer tendências ou padrões observados nos resultados de teste.

Histórico de Teste: Manter um histórico de testes ao longo do tempo é importante para acompanhar o progresso do teste, identificar tendências de qualidade e avaliar a eficácia das estratégias de teste ao longo do ciclo de vida do projeto.

44

DOCUMENTAÇÃO DE IMPLANTAÇÃO E OPERAÇÃO

A documentação na fase de implantação e operação do software é essencial para garantir uma transição suave do ambiente de desenvolvimento para o ambiente de produção e para garantir a operação eficaz e eficiente do software após sua implantação.

45

DOCUMENTAÇÃO DE IMPLANTAÇÃO E OPERAÇÃO

Procedimentos de Implantação: A documentação de implantação descreve os procedimentos necessários para instalar e configurar o software em um ambiente de produção.

Configuração do Ambiente: A documentação de implantação também descreve os requisitos de hardware e software necessários para executar o software, bem como as configurações específicas do ambiente, como configurações de rede, bancos de dados e servidores de aplicativos.

46

DOCUMENTAÇÃO DE IMPLANTAÇÃO E OPERAÇÃO

Guia de Operações: Após a implantação do software, a documentação fornece um guia de operações que descreve como operar e manter o software em um ambiente de produção.

Gestão de Configuração: A documentação de implantação pode incluir informações sobre gestão de configuração, que descreve como controlar e gerenciar as configurações do sistema ao longo do tempo.

47

DOCUMENTAÇÃO DE IMPLANTAÇÃO E OPERAÇÃO

Políticas de Segurança: A documentação de implantação também deve abordar políticas de segurança e práticas recomendadas para proteger o sistema contra ameaças de segurança.

Manuais do Usuário: Para sistemas que serão usados por usuários finais, a documentação de implantação pode incluir manuais do usuário que descrevem como usar o software de forma eficaz e eficiente.

48

DOCUMENTAÇÃO DO USUÁRIO

A documentação do usuário é uma parte essencial do processo de desenvolvimento de software, pois fornece informações detalhadas sobre como usar o sistema de forma eficaz e eficiente.

49

DOCUMENTAÇÃO DO USUÁRIO

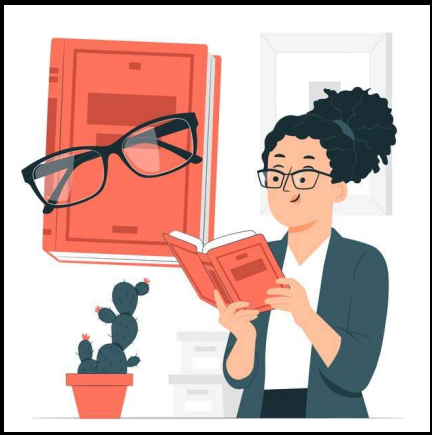
Manuais do Usuário: Os manuais do usuário são documentos escritos que fornecem instruções passo a passo sobre como usar o software.

Guias de Início Rápido: Os guias de início rápido são documentos resumidos que ajudam os usuários a começarem a usar o software rapidamente.

Tutoriais Interativos: Alguns sistemas incluem tutoriais interativos que guiam os usuários através das principais funcionalidades do software, passo a passo.

50

DOCUMENTAÇÃO DO USUÁRIO



FAQs (Perguntas Frequentes): Os FAQs são documentos que respondem a perguntas comuns dos usuários sobre o software.

Vídeos e Demonstrativos: Além de documentação escrita, os vídeos e demonstrativos também podem ser utilizados para fornecer instruções visuais sobre como usar o software.

Suporte ao Cliente: Além da documentação escrita, é importante fornecer canais de suporte ao cliente, como chats ao vivo, fóruns de discussão e sistemas de tickets de suporte.

Atualizações de Documentação: À medida que o software é atualizado e novas funcionalidades são adicionadas, a documentação do usuário deve ser atualizada para refletir essas mudanças.

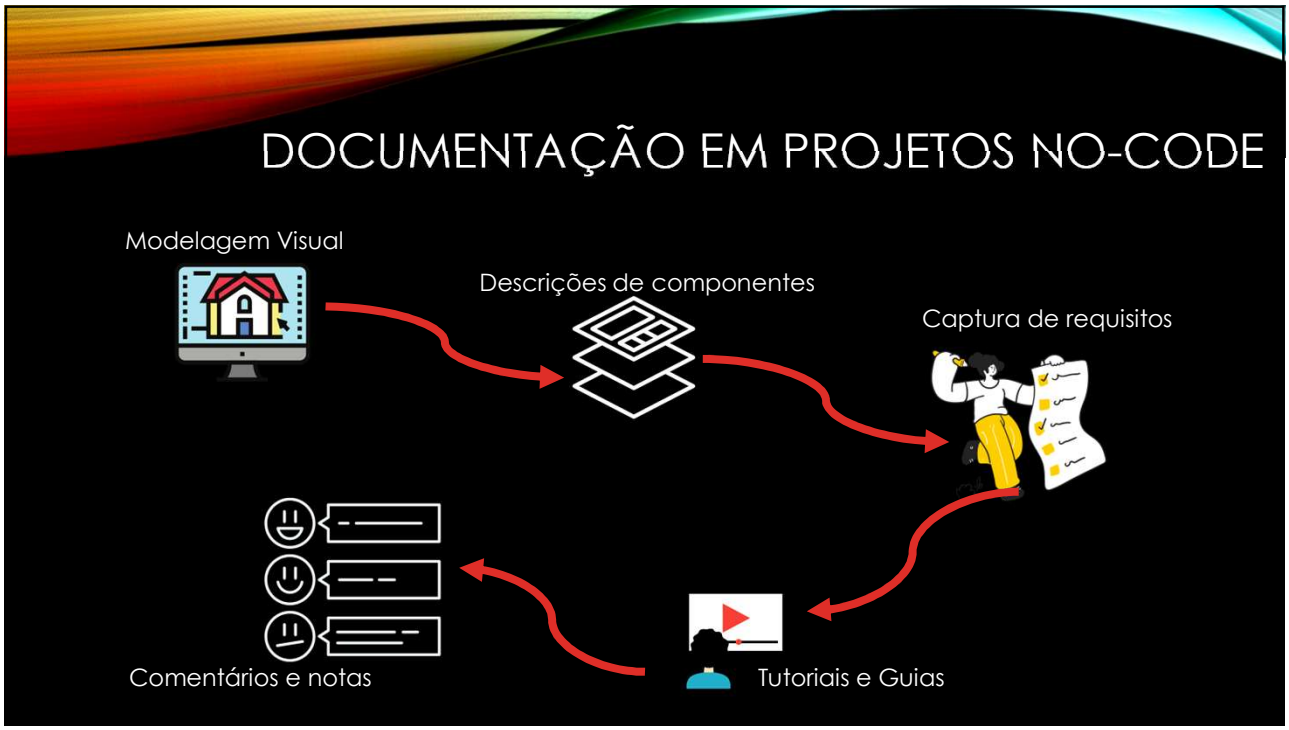
51

DOCUMENTAÇÃO EM PROJETOS NO-CODE

Em projetos No-Code, a documentação desempenha um papel crucial na captura e comunicação dos requisitos, design e funcionamento do sistema, apesar da ausência de código tradicional.

Veja o passo a passo da documentação em projetos no-code

52


DOCUMENTAÇÃO EM PROJETOS NO-CODE
Modelagem Visual
Descrições de componentes
Captura de requisitos
Comentários e notas
Tutoriais e Guias