

ADS - Engenharia de Software 2025 - Anotações  
de aula

Professor Miguel Suez Xve Penteado

2025-04-08



# Contents

|  |           |
|--|-----------|
| <b>Sobre estas anotações</b>   | <b>7</b>  |
| <b>INTRODUÇÃO A DISCIPLINA DE ENGENHARIA DE SOFTWARE</b>             | <b>15</b> |
| 0.1 O que é ENGENHARIA DE SOFTWARE . . . . .                         | 15        |
| <b>1 QUALIDADE DE SOFTWARE</b>                                       | <b>17</b> |
| 1.1 COMPLIANCE . . . . .   | 17        |
| 1.2 QUALIDADE . . . . .  | 18        |
| 1.3 Exercício de Fixação: . . . . .                                  | 30        |
| <b>2 Verificação e Validação de Software</b>                         | <b>37</b> |
| 2.1 Verificação de Software: . . . . .                               | 38        |
| 2.2 Validação de Software: . . . . .                                 | 38        |
| 2.3 Classificação das Técnicas de Verificação e Validação: . . . . . | 39        |
| 2.4 Verificação e Validação de software por Técnicas Estáticas . . . | 39        |
| 2.5 Verificação e Validação de software por Técnicas Dinâmicas . .   | 42        |
| 2.6 Exercícios . . . . .   | 44        |
| <b>3 Verificação e Validação de Software II - Continuação</b>        | <b>51</b> |
| 3.1 Fundamentos de Teste de Software . . . . .                       | 51        |
| 3.2 Os testes e o Ciclo de Vida do Software . . . . .                | 51        |
| 3.3 Modelo V . . . . .   | 55        |
| 3.4 Testes Unitários . . . . .                                       | 56        |

|          |   |           |
|----------|---|-----------|
| 3.5      | Exercícios . . . . .  | 58        |
| 3.6      | Cadastro de Clientes . . . . .  | 66        |
| 3.7      | Cadastro de Fornecedores . . . . .  | 67        |
| 3.8      | Cadastro de Produtos . . . . .  | 67        |
| <b>4</b> | <b>Introdução à Manutenção de Software</b>                                | <b>69</b> |
| 4.1      | 1- Manutenção: definição e características” . . . . .                     | 71        |
| 4.2      | 2- Introdução à Manutenibilidade . . . . .                                | 75        |
| 4.3      | Exercícios . . . . .  | 75        |
| <b>5</b> | <b>APROFUNDANDO A MANUTENIBILIDADE E AS TÉCNICAS DE DESENVOLVIMENTO</b>   | <b>77</b> |
| 5.1      | Manutenibilidade . . . . .  | 77        |
| 5.2      | Técnicas de Desenvolvimento para a Manutenibilidade . . . . .             | 78        |
| <b>6</b> | <b>PROCESSOS E PADRÕES NA MANUTENÇÃO DE SOFTWARE</b>                      | <b>79</b> |
| 6.1      | Processos de Manutenção . . . . .   | 79        |
| 6.2      | Padrões de Desenvolvimento . . . . .                                      | 80        |
| 6.3      | Padrões de Manutenção . . . . .   | 80        |
| <b>7</b> | <b>ABORDAGENS MODERNAS E ATIVIDADES DE APOIO À MANUTENÇÃO</b>             | <b>81</b> |
| 7.1      | Desenvolvimento Baseado em Componentes e Impactos na Manutenção . . . . . | 81        |
| 7.2      | Desenvolvimento Orientado a Aspectos e Impactos na Manutenção             | 82        |
| 7.3      | Atividades de Apoio a Manutenção . . . . .                                | 82        |
| <b>8</b> | <b>Gerência de Configuração</b>   | <b>83</b> |
| 8.1      | Introdução à Gerência de Configuração . . . . .                           | 83        |
| 8.2      | Elementos da Gerência de Configuração . . . . .                           | 84        |
| 8.3      | Processo de Gerência de Configuração . . . . .                            | 85        |
| 8.4      | Ferramentas de Gerência de Configuração . . . . .                         | 86        |
| 8.5      | GC em Contextos Ágeis e Tradicionais . . . . .                            | 86        |
| 8.6      | Referências: . . . . .  | 86        |

|  |            |
|--|------------|
| <b>CONTENTS</b>                                | <b>5</b>   |
| <b>9 Revisão para NP2</b>                      | <b>87</b>  |
| <b>10 Revisão para a Substitutiva</b>          | <b>89</b>  |
| <b>11 Referencias</b>                          | <b>91</b>  |
| <b>12 Apêndice I - Estudo da - ERP Agrotec</b> | <b>93</b>  |
| 12.1 Entrega #01 - Módulo Cadastros . . . . .  | 93         |
| 12.2 Interface JanelaPrincipal . . . . .       | 93         |
| 12.3 Cadastro de Clientes . . . . .            | 97         |
| 12.4 Cadastro de Fornecedores . . . . .        | 98         |
| 12.5 Cadastro de Produtos . . . . .            | 98         |
| <b>13 Apendice II</b>                          | <b>99</b>  |
| <b>14 Apendice III</b>                         | <b>101</b> |



# **Sobre estas anotações**

Estas anotações são apenas lembretes das aulas expostas em sala, durante a disciplina de ENGENHARIA DE SOFTWARE.

**0.0.1 ACESSO AO GITBOOK CELULAR****0.0.2 <https://miguel7penteado.github.io/ADS-EngenhariaSoftware2025>**

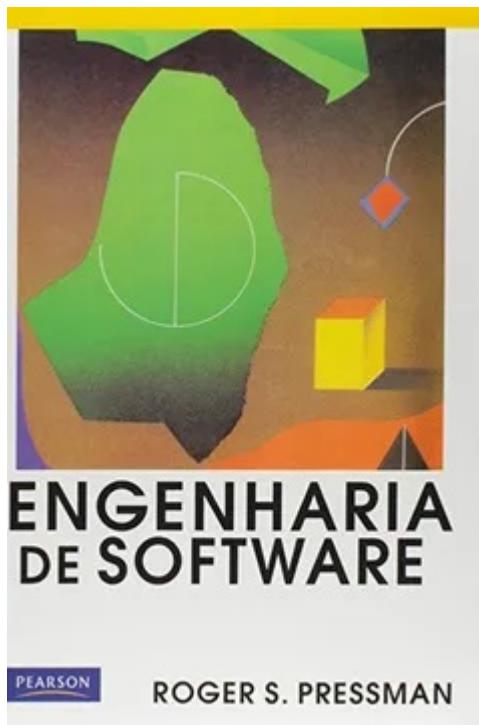
0.0.3 APP EPUB ANDROID

0.0.4 Moon+ Reader



### 0.0.5 Livros Texto da Disciplina

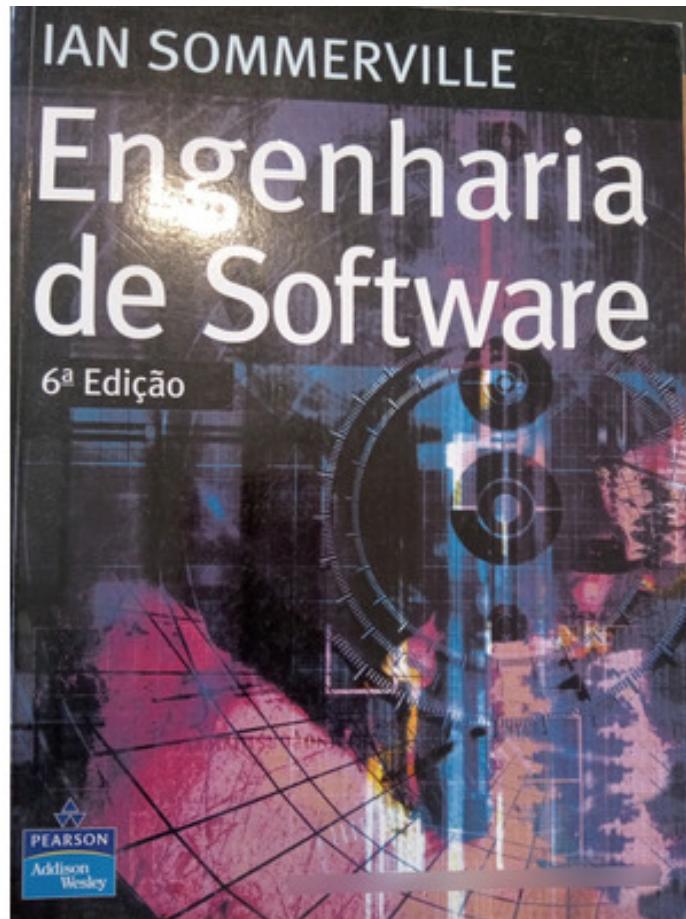
#### 0.0.5.1 “Engenharia de Software” do autor “Roger S Pressman”



---

|                   |                          |
|-------------------|--------------------------|
| Autor(es)         | Roger S. Pressman        |
| Editora           | Pearson                  |
| Idioma            | Português                |
| ISBN              | 8534602379 9788534602372 |
| Formato           | Capa comum               |
| Páginas           | 1056                     |
| Código Biblioteca |                          |

---

**0.0.5.2 “Engenharia de Software” do autor “Ian Sommerville”**

---

|                          |                 |
|--------------------------|-----------------|
| <b>Autor(es)</b>         | Ian SommerVille |
| <b>Editora</b>           | Pearson         |
| <b>Idioma</b>            | Português       |
| <b>ISBN</b>              | 9788588639072   |
| <b>Formato</b>           | Capa comum      |
| <b>Páginas</b>           | 768             |
| <b>Código Biblioteca</b> |                 |

---

Calendário das aulas

**0.0.5.2.1 FEVEREIRO 2025**

---

| Data            | Dia da semana | Aulas                   |
|-----------------|---------------|-------------------------|
| 4 de fevereiro  | Terça-feira   | Recesso                 |
| 11 de fevereiro | Terça-feira   | Recesso                 |
| 18 de fevereiro | Terça-feira   | Aula Inaugural          |
| 25 de fevereiro | Terça-feira   | Qualidade de Software I |

---

#### 0.0.5.2.2 MARÇO 2025

---

| Data        | Dia da semana | Aulas                                  |
|-------------|---------------|--|
| 4 de março  | Terça-feira   | Carnaval                               |
| 11 de março | Terça-feira   | Verificação e Validação de Software I  |
| 18 de março | Terça-feira   | Verificação e Validação de Software II |
| 25 de março | Terça-feira   |  |

---

#### 0.0.5.2.3 ABRIL DE 2025

---

| Data        | Dia da semana | Aulas                      |
|-------------|---------------|----------------------------|
| 1 de abril  | Terça-feira   | Prova NP1                  |
| 8 de abril  | Terça-feira   | Manutenção de software I   |
| 15 de abril | Terça-feira   | Manutenção de software II  |
| 22 de abril | Terça-feira   | Manutenção de software III |
| 29 de abril | Terça-feira   | Manutenção de software IV  |

---

#### 0.0.5.2.4 MAIO DE 2025

---

| Data       | Dia da semana | Aulas                    |
|------------|---------------|--------------------------|
| 6 de maio  | Terça-feira   | Gerência de Configuração |
| 13 de maio | Terça-feira   | Revisão                  |
| 20 de maio | Terça-feira   | Prova NP2                |
| 27 de maio | Terça-feira   | Substitutiva             |

---

#### 0.0.5.2.5 JUNHO DE 2025

---

| Data        | Dia da semana | Aulas   |
|-------------|---------------|---------|
| 3 de junho  | Terça-feira   | Plantão |
| 10 de junho | Terça-feira   | Plantão |
| 17 de junho | Terça-feira   | Exame   |
| 24 de junho | Terça-feira   |         |

---

## 0.0.6 Alunos 2025

### 0.0.6.1 Turma DS2P40

| Matrícula | Nome do Aluno                |
|-----------|------------------------------|
| F35HFJ-1  | BEATRIZ ALMEIDA DA SILVA     |
| R54885-6  | BRENO SOUZA MASCARENHAS      |
| R19267-9  | CARLOS EDUARDO DA S GALDINO  |
| R150FH-8  | DANILO LUCAS LOURENCO        |
| G740IF-9  | GUSTAVO ALCANTARA NOBRE      |
| G76IBD-7  | HELLEN REGINA B DOS SANTOS   |
| F35EBD-4  | JOAO ALFREDO DA S BRENNER    |
| R11835-5  | LUCAS ROSSE                  |
| G839GC-6  | PABLO HENRIQUE C ARAUJO      |
| G61ICI-3  | THIAGO VERNIER LOUREIRO MAIA |

### 0.0.6.2 Turma DS3P40

| Matrícula | Nome do Aluno                  |
|-----------|--------------------------------|
| T736DG-3  | ANDRE LUIS RIGUEIRA ZANA       |
| R06534-0  | BIANCA CAVALCANTE DOS SANTOS   |
| G964AA-5  | CIBELE MARIA BARBOSA           |
| R1007A-0  | DANIEL GOES CARVALHO SILVA     |
| G98399-8  | DAVI PEREIRA DO VALE           |
| F3567F-6  | EDUARDO MONTINO LACERDA        |
| F35973-5  | FELIPE DE CAMPOS MOREIRA ALVES |
| R0622A-5  | FERNANDA VICTORIA D LO VACCO   |
| R091EC-3  | GABRIEL ALVES BATISTA          |
| G989DC-6  | GABRIEL PINHEIRO SOUZA         |
| R08565-1  | GIOVANNY GOMES BRANDAO         |
| R055AF-2  | GUILHERME NASCIMENTO R BARBOSA |
| N088EG-0  | GUILHERME R DE OLIVEIRA        |
| R06229-5  | GUSTAVO SILVA DOS SANTOS       |
| R07095-6  | HENRIQUE MOREIRA BOTELLA       |
| R0823C-0  | HENRIQUE P DOS S FRANCISCO     |
| G98BGB-2  | IGOR XAVIER DE MATTOS          |
| G90JDE-2  | JOAO RICARDO DA SILVA JUNIOR   |
| F3590G-2  | JOAO RICK GALDINO PEREIRA      |
| R0567D-6  | JOAO VICTOR CARVALHO DE SOUZA  |
| G9756E-6  | JOAO VICTOR DA SILVA MACHADO   |
| G0249I-6  | JULIANA BORGES MOURA           |
| F35937-9  | MATHEUS SERVULO CAJE           |

| Matrícula | Nome do Aluno                  |
|-----------|--------------------------------|
| R10099-5  | MELISSA SCARPINATTI B DA SILVA |
| G8832G-1  | RENAN PRAZERES CLEMENTINO      |
| F35CDF-2  | SERGIO ALEXANDRE A DO AMARAL   |

#### 0.0.6.3 Turma DS3Q40

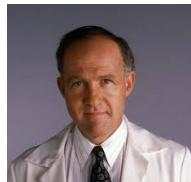
Com base nas informações da fonte “**DS3Q40.pdf**” e em nossa conversa anterior, apresento novamente a tabela com a coluna **Matrícula** (RA) e **Nome do Aluno** da turma **DS3Q40**:

| Matrícula | Nome do Aluno                  |
|-----------|--------------------------------|
| G003II-9  | ALEX LIMA SILVA                |
| G0327I-4  | AMANDA SIMONETTO DIAS          |
| G02JDI-5  | ATILA WILLIAM F DE BARROS      |
| R096DH-9  | BRENDA RUOTTI                  |
| R0087I-2  | GUSTAVO SILVA DE ARAUJO        |
| G99JAH-4  | JESSICA SANTOS ANJOS           |
| G8811G-1  | KAIKY ALVES MONTEIRO           |
| G99319-5  | KLEBER WENDEL DE ALMEIDA RIBAS |
| G90EJA-1  | LEONARDO OLIVEIRA DOS SANTOS   |
| G99ACJ-8  | LUCAS SILVA PINTO DE ASSIS     |
| G99843-0  | MATHEUS ALVES LIMA             |
| G996FJ-4  | MATHEUS DE OLIVEIRA MONTEIRO   |
| G99JFJ-7  | MATHEUS RIBEIRO DE CAMPOS      |
| G9931A-5  | PEDRO HENRIQUE CAMPOS LEAL     |
| G012IF-3  | PEDRO PAULO VITALINO           |
| R094GC-7  | RENAN DOS SANTOS FERREIRA      |
| G96JFG-6  | RICHARD TRISTAN P GARCIA       |
| G92GHH-8  | RODRIGO SANTOS ARAUJO          |
| G977HG-0  | SIDNEI SERRAO DA SILVA         |
| G003IC-0  | THIAGO DA SILVA SEIXEIRO       |
| G99566-0  | YASMIN HELENA DE OLIVEIRA FERN |

# INTRODUÇÃO A DISCIPLINA DE ENGENHARIA DE SOFTWARE

Do que trata esta disciplina e o que quer dizer o termo que dá nome a ela ?

## 0.1 O que é ENGENHARIA DE SOFTWARE



**Engenharia de Software** é o processo de desenvolvimento de programas de computador, estruturas de dados e documentos. (**Roger S. Pressman**)



**Engenharia de Software** é uma disciplina de engenharia que se preocupa com todo o processo de produção de software. Isso inclui desde a especificação do sistema até a sua manutenção. (**Ian Sommerville**)

É atribuído a **Margaret Hamilton**, desenvolvedora do programa de navegação da APOLLO 11 a criação do termo ENGENHARIA DE SOFTWARE.



# **Chapter 1**

## **QUALIDADE DE SOFTWARE**

### **1.1 COMPLIANCE**

Para que uma organização consiga fechar contrados de venda ou fornecimento com outra organização, especialmente quando o valor do contrato de venda ou prestação é muito alto, há um processo de checagem de COMPLIANCE:

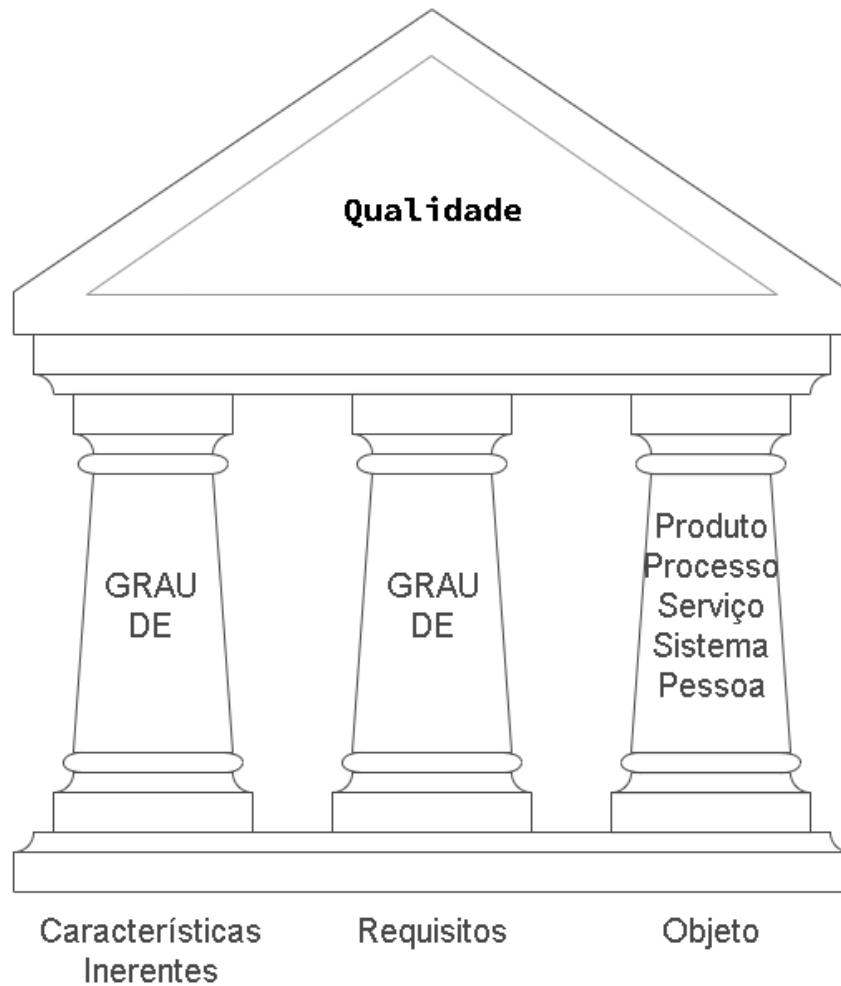


## 1.2 QUALIDADE

O que é Qualidade ? (Definição ISO 9000)

Qualidade é definida como o grau em que um conjunto de características inerentes de um objeto satisfaz requisitos onde: **Características inerentes** São propriedades que fazem parte do objeto, onde:

- **Requisitos:** São as necessidades ou expectativas declaradas, geralmente implícitas ou obrigatórias;
- **objeto** pode ser representado por um produto, serviço, processo, organização, sistema ou pessoa;



### 1.2.1 QUALIDADE APLICADA A PRODUTO

O CONTROLE DE QUALIDADE do PRODUTO concentra-se em aperfeiçoar:

- as **características** e
- o **desempenho** do produto em si,

visando atender às necessidades e expectativas dos clientes.

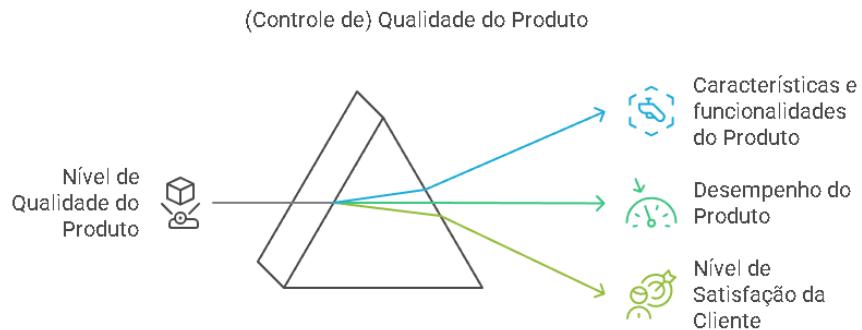


Table 1.1: Resultado esperado do CONTROLE DE QUALIDADE aplicado ao PRODUTO

---

#### Resultados do CONTROLE DE QUALIDADE aplicado ao PRODUTO

---

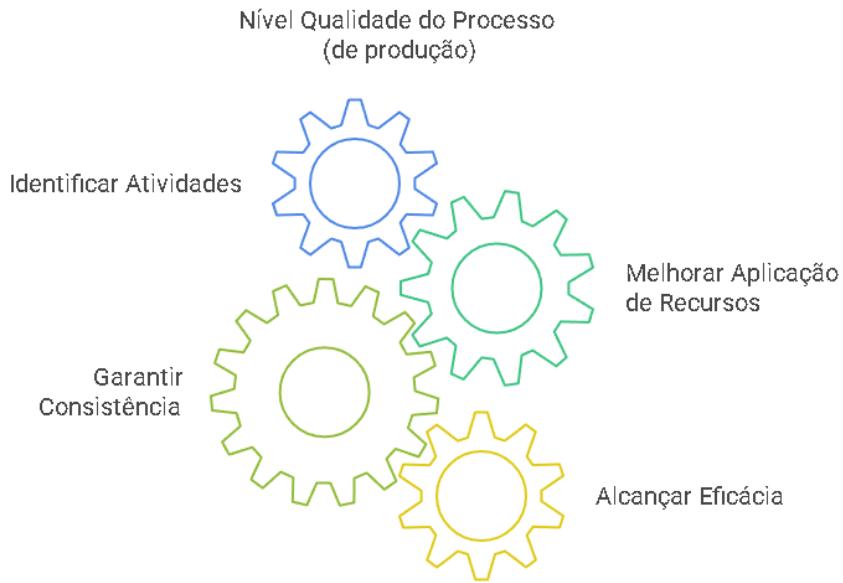
- Aumento no GRAU das características e funcionalidades do produto.
  - Aumento no GRAU de desempenho do produto.
  - Aumento no GRAU de nível de satisfação do cliente.
- 

### 1.2.2 QUALIDADE APLICADA A PROCESSO

O CONTROLE DE QUALIDADE DE PROCESSO concentra-se em aperfeiçoar

- as **atividades** e
- melhor **aplicação dos recursos**

utilizados para criar o produto, visando garantir a consistência e a eficácia da produção.



---

Resultados do CONTROLE DE QUALIDADE aplicado ao  
PROCESSO

---

- **identificar** as ATIVIDADES do processo.
  - **Garantir a Consistência** as ATIVIDADES do processo.
  - **Melhorar** a APLICAÇÃO DE RECURSOS do processo.
  - **Alcançar** a **EFICÁCIA**.
- 

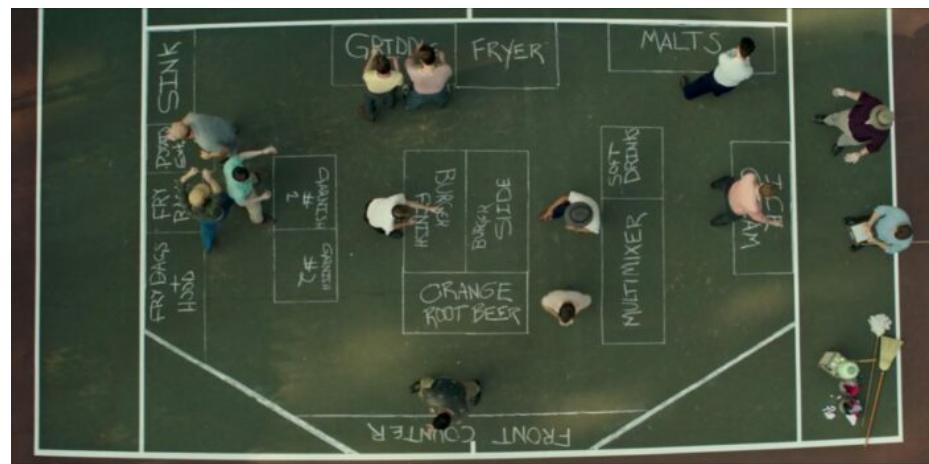
### 1.2.3 CASO MACDONALDS - Qualidade de Produto e Processo

O filme “Fome de Poder” (“The Founder”, no original) narra a história real da ascensão da rede McDonald’s, desde sua origem como uma pequena hamburgheria na Califórnia até se tornar um império global do fast-food.

- Reconhecimento da **qualidade do produto** - hamburguers McDonalds



Reconhecimento da **Qualidade do Processo** de fabricação do Produto



A Jornada de criação da rede de Franquias Mc Donald's por Ray Kroc



- Reconhecimento da Capacidade de Franquia (Replicação):



#### 1.2.4 QUALIDADE NAS ORGANIZAÇÕES

#### 1.2.5 Família ISO 9000

A **NBR ISO 9000** é um conjunto de normas técnicas que estabelecem diretrizes e padrões para a criação de um **Sistema de Gestão da Qualidade (SGQ)**.

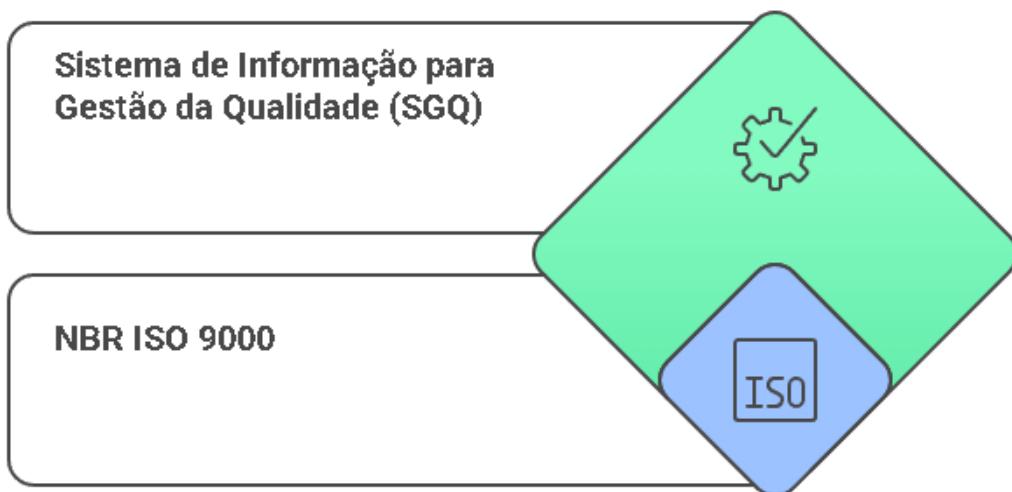
O sistema SGQ (um si que pode ou não ser um pacote de software) deve mapear

---

| Áreas<br>mapeadas<br>por um<br>sistema SGQ | PROCESSOS | POLÍTICAS | PROCEDIMENTOS | RESPONSABILIDADES |
|--|-----------|-----------|---------------|-------------------|
|--|-----------|-----------|---------------|-------------------|

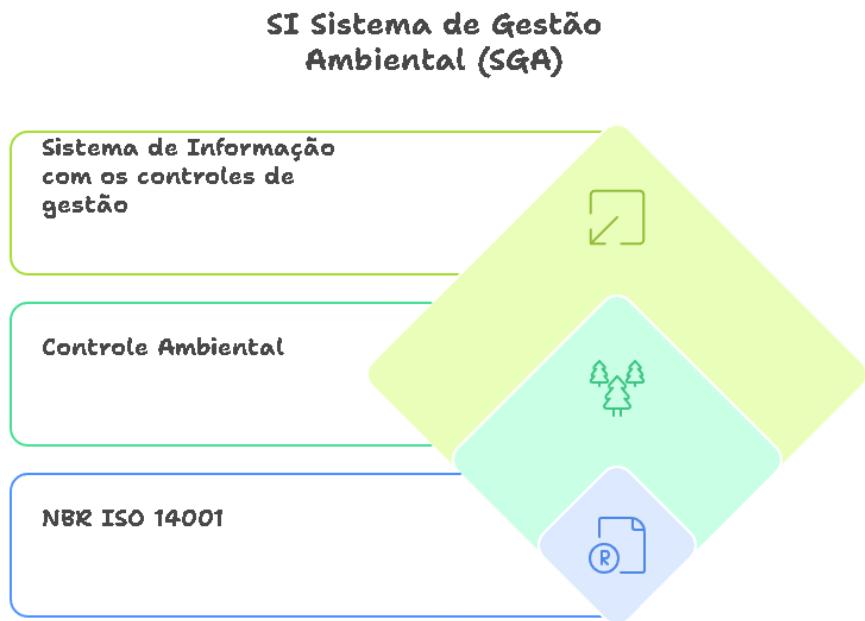
---

**SI Sistema de Gestão da Qualidade (SGQ)  
Genérico**



#### 1.2.6 Família ISO 14000

A **NBR ISO 14000** é um conjunto de normas técnicas que tratam de GESTÃO AMBIENTAL nas organizações. Estabelecem normas e diretrizes para criar (SI) Sistemas de Gestão Ambiental (SGA):

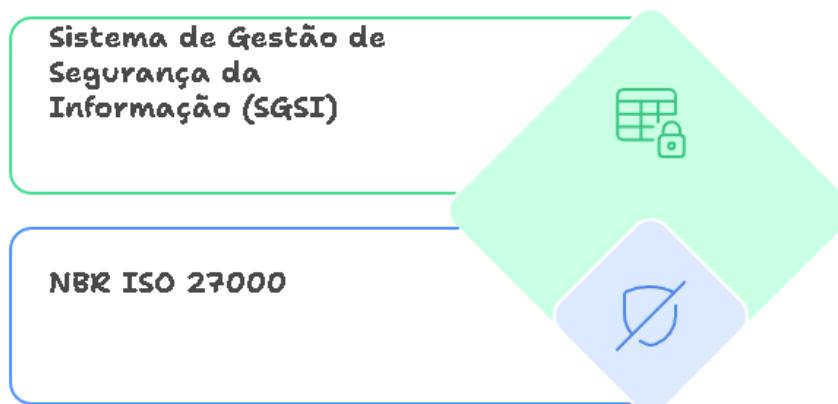


### 1.2.7 Família ISO 27000

**NBR ISO 27000**, trata de normas para **gestão segurança da Informação**. Fornecem um framework para a gestão da segurança da informação em organizações.

Especifica os requisitos para a criação de um(SI) Sistema de Gestão de Segurança da Informação (SGSI).

## Sistema de Gestão de Segurança da Informação (SGSI)



### 1.2.8 Segmentos das Organizações e Adoção das normas de Qualidade



### 1.2.9 QUALIDADE NA ENGENHARIA DE SOFTWARE

A qualidade de software não define S.I.s

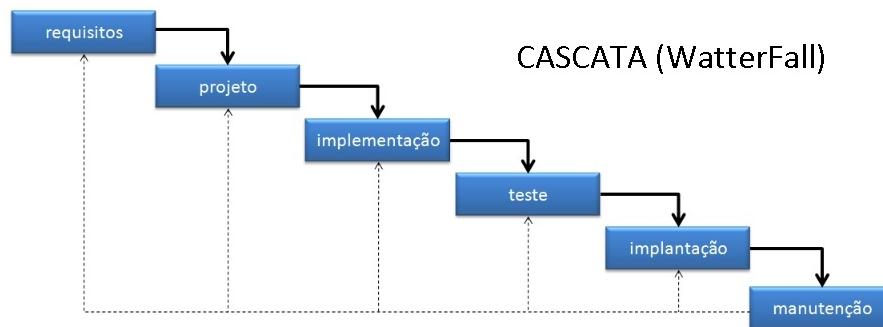
### 1.2.10 Família NBR ISO 9126

Focava na qualidade do produto de software, definindo um conjunto de parâmetros para padronizar a avaliação dessa qualidade. Ela se enquadrava no modelo de qualidade das normas da família 9000.



### 1.2.11 Família NBR ISO 12207

A norma ISO 12207 define um conjunto de processos para o ciclo de vida do software. Seu principal foco é estabelecer um framework padronizado para o desenvolvimento, manutenção e descarte de software, visando garantir a qualidade e a eficiência desses processos.

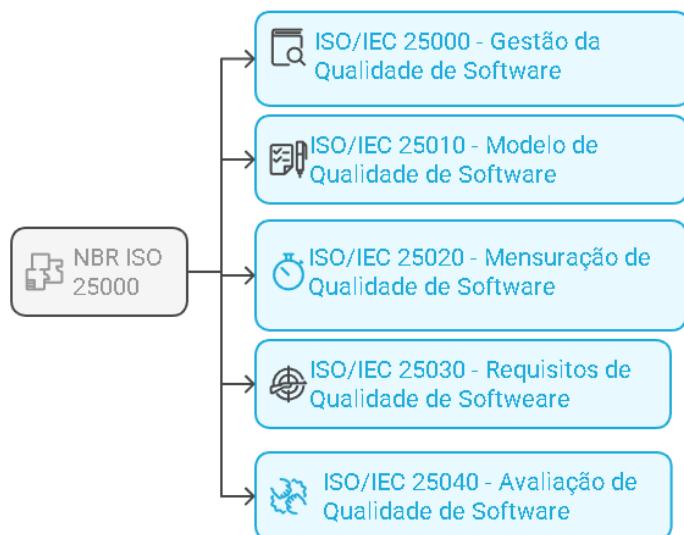




### 1.2.12 Família NBR ISO 25000

A **NBR ISO 25000**, também conhecida como SQuaRE (Software Product Quality Requirements and Evaluation - Requisitos e Avaliação da Qualidade de Produtos de Software), é uma série de normas internacionais que fornecem um subconjunto de normas para a avaliação da qualidade de produtos de software. Este subconjunto é formado pelas normas **ISO/IEC 25000**, **ISO/IEC 25010**, **ISO/IEC 25020**, **ISO/IEC 25030** e **ISO/IEC 25040**.

NBR ISO 25000 - SQuaRE -  
Software Product Quality  
Requirements and  
Evaluation



### 1.3 Exercício de Fixação:

#### 1.3.1 Testes:

---

TESTE 01

**Qual das seguintes alternativas melhor descreve o conceito de qualidade, de acordo com a definição apresentada?**

- a) Qualidade é a ausência de defeitos em um produto ou serviço.
- b) Qualidade é o grau em que um produto ou serviço excede as expectativas do cliente.

---

**TESTE 01**

- c) Qualidade é o grau em que um conjunto de características inerentes de um objeto satisfaz requisitos.
  - d) Qualidade é a conformidade com as normas e regulamentos estabelecidos.
  - e) Qualidade é a capacidade de um produto ou serviço ser produzido em grande quantidade.
- 

---

**TESTE 02**

**Qual das seguintes alternativas melhor descreve o conceito de Compliance?**

- a) Compliance é um conjunto de estratégias de marketing para aumentar a visibilidade da empresa.
  - b) Compliance é um sistema de gestão financeira para otimizar os lucros da empresa.
  - c) Compliance é o conjunto de normas, procedimentos e práticas para cumprir legislação e padrões éticos, visando segurança e minimização de riscos.
  - d) Compliance é um programa de treinamento para melhorar o desempenho dos funcionários.
  - e) Compliance é uma ferramenta de análise de mercado para identificar oportunidades de negócio.
- 

---

**TESTE 03**

**Qual das seguintes alternativas melhor descreve o conceito de qualidade aplicada ao produto?**

- a) Qualidade do produto é a capacidade de um produto ser vendido a um preço baixo.
  - b) Qualidade do produto é o grau em que um produto atende às expectativas do cliente em relação às suas características inerentes e o desempenho.
  - c) Qualidade do produto é a quantidade de produtos produzidos em um determinado período de tempo.
  - d) Qualidade do produto é a aparência estética de um produto, independentemente de sua funcionalidade.
  - e) Qualidade do produto é a capacidade de um produto ser facilmente descartado após o uso.
-

**TESTE 04**

**Qual das seguintes alternativas melhor descreve o conceito de qualidade aplicada ao processo?**

- a) Qualidade no processo se refere à inspeção final do produto para garantir que ele esteja livre de defeitos.
  - b) Qualidade no processo é a capacidade de um processo produzir resultados consistentes e previsíveis, atendendo aos requisitos estabelecidos.
  - c) Qualidade no processo é a utilização de materiais de alta qualidade na fabricação do produto.
  - d) Qualidade no processo é a implementação de um sistema de gestão da qualidade certificado, como a ISO 9001.
  - e) Qualidade no processo é a satisfação do cliente com o produto final, independentemente de como ele foi produzido.
- 

**TESTE 05**

**Quais das alternativas melhor reflete principais resultados do controle de qualidade aplicado ao produto?**

- a) Aumento no grau das características e funcionalidades do produto, aumento no grau de desempenho do produto e aumento no grau de nível de satisfação do cliente.
  - b) Redução de custos de produção, aumento da eficiência dos processos e diminuição do tempo de entrega.
  - c) Melhoria na imagem da empresa, aumento da participação de mercado e expansão para novos mercados.
  - d) Padronização dos produtos, simplificação dos processos de fabricação e redução do desperdício de materiais.
  - e) Maior flexibilidade na produção, personalização dos produtos e aumento da variedade de produtos oferecidos.
- 

**TESTE 06**

**Qual era o principal objetivo da família de normas ISO/NBR 9126?**

- a) Definir padrões para a gestão de projetos de software.
- b) Estabelecer diretrizes para a segurança da informação em sistemas de software.

---

**TESTE 06**

- c) Padronizar a documentação de software e os processos de desenvolvimento.
  - d) Promover a interoperabilidade entre diferentes sistemas de software.
  - e) Padronizar a avaliação da qualidade de produtos de software, definindo parâmetros para essa avaliação dentro do modelo de qualidade das normas da família 9000.
- 

---

**TESTE 07**

Qual tipo de Sistema de Informação (SI) a família de normas ISO 9000 propunha implementar?

- a) Sistema de Gestão Financeira (SGF)
  - b) Sistema de Gestão de Recursos Humanos (SGRH)
  - c) Sistema de Gestão da Qualidade (SGQ)
  - d) Sistema de Gestão de Produção (SGP)
  - e) Sistema de Gestão de Marketing (SGM)
- 

---

**TESTE 08**

Qual área a família de normas ISO 14000 trata e qual tipo de Sistema de Informação (SI) ela propõe implementar?

- a) Trata da gestão da qualidade e propõe implementar um Sistema de Gestão da Qualidade (SGQ).
  - b) Trata da segurança da informação e propõe implementar um Sistema de Gestão de Segurança da Informação (SGSI).
  - c) Trata da gestão de projetos e propõe implementar um Sistema de Gestão de Projetos (SGP).
  - d) Trata da gestão de recursos humanos e propõe implementar um Sistema de Gestão de Recursos Humanos (SGRH).
  - e) Trata da gestão ambiental e propõe implementar um Sistema de Gestão Ambiental (SGA).
- 

---

**TESTE 09**

Qual é o principal objetivo da norma ISO 12207?

**TESTE 09**

- a) Definir um conjunto de processos para o ciclo de vida do software, estabelecendo um framework padronizado para desenvolvimento, manutenção e descarte, visando qualidade e eficiência.
  - b) Estabelecer diretrizes para a segurança da informação em sistemas de software.
  - c) Padronizar a documentação de software e os processos de desenvolvimento ágil.
  - d) Promover a interoperabilidade entre diferentes sistemas de software e hardware.
  - e) Definir padrões para a gestão de projetos de software, focando na otimização de custos e prazos.
- 

**TESTE 10**

**Qual é o principal objetivo da família de normas NBR ISO 25000 (SQuaRE)?**

- a) Definir padrões para a gestão de projetos de software.
  - b) Estabelecer diretrizes para a segurança da informação em sistemas de software.
  - c) Padronizar a avaliação da qualidade de produtos de software.
  - d) Promover a interoperabilidade entre diferentes sistemas de software.
  - e) Definir processos para o ciclo de vida do software.
- 

**TESTE 11**

Qual subnorma da NBR ISO 25000 (SQuaRE) é responsável por definir modelos de qualidade para produtos de software?

Alternativas:

- a) ISO/IEC 25020
  - b) ISO/IEC 25030
  - c) ISO/IEC 25040
  - d) ISO/IEC 25010
  - e) ISO/IEC 25000
-

---

TESTE 12

---

Qual subnorma da NBR ISO 25000 (SQuaRE) fornece diretrizes para a avaliação da qualidade de produtos de software?

- a) ISO/IEC 25040
  - b) ISO/IEC 25020
  - c) ISO/IEC 25030
  - d) ISO/IEC 25010
  - e) ISO/IEC 25000
- 

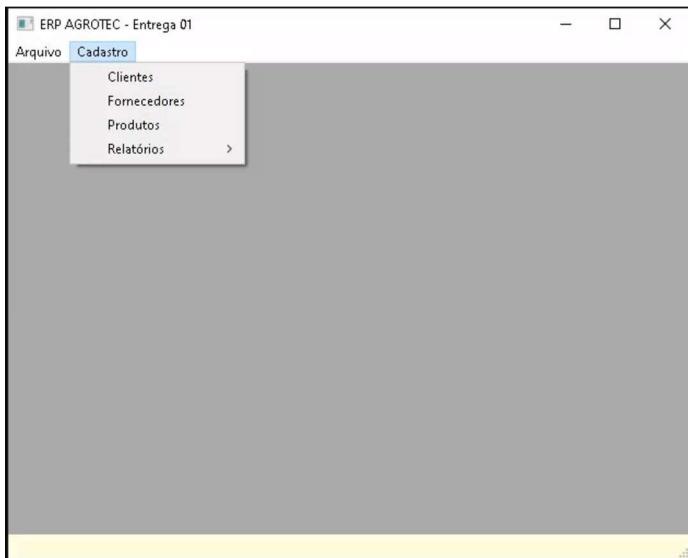
### 1.3.2 Repostas dos testes

| Teste | Alternativa | Correta |
|-------|-------------|---------|
| 01    | c           |         |
| 02    | c           |         |
| 03    | b           |         |
| 04    | b           |         |
| 05    | a           |         |
| 06    | e           |         |
| 07    | c           |         |
| 08    | e           |         |
| 09    | a           |         |
| 10    | c           |         |
| 11    | d           |         |
| 12    | a           |         |



## Chapter 2

# Verificação de Validação de Software



Conforme sabemos existem quatro **atividades** fundamentais no **processo de engenharia de software**. Estas atividades podem ser organizadas de diferentes maneiras dependendo do processo de desenvolvimento utilizado. A seguinte tabela resume as atividades do processo de software de acordo com Sommerville:

| Atividade do Processo de Software          | Descrição   |
|--|---|
| <i>Especificação de Software</i>           | <i>A funcionalidade do software e as restrições ao seu funcionamento devem ser definidas.</i>   |
| <i>Projeto e Implementação de Software</i> | <i>O software deve ser produzido para atender às especificações.</i>  |
| <b>Validação de Software</b>               | <b>O software deve ser validado para garantir que atenda às demandas do cliente.</b>  |
| <i>Evolução de Software</i>                | <i>O software deve evoluir para atender às necessidades de mudança dos clientes. Alterações no software são uma parte inevitável.</i> |

Hoje vamos explorar a terceira etapa, mas especificamente a *Verificação e Validação de Software*.



## 2.1 Verificação de Software:

**Definição de Verificação de Software:** Assegurar que o software implementa corretamente uma função específica. “Estamos criando o produto corretamente?”.

## 2.2 Validação de Software:

**Definição de Validação de Software:** Assegurar que o software foi criado e pode ser rastreado segundo os requisitos do cliente. “Estamos criando o produto certo?”. Validação tem sucesso quando o

### **2.3. CLASSIFICAÇÃO DAS TÉCNICAS DE VERIFICAÇÃO E VALIDAÇÃO:39**

*software funciona de uma maneira que pode ser razoavelmente esperada pelo cliente.*

Quais os objetivos globais da etapa de Verificação e Validação de Software ?

---

#### **Objetivos Globais - Etapa de Verificação e Validação do Processo de Software**

---

- 1) Conscientizar sobre a importância da V&V para a qualidade do software produzido.
  - 2) Identificar erros precocemente.
  - 3) Reduzir os custos de desenvolvimento do software.
  - 4) Assegurar que o software atenda aos requisitos do cliente.
- 

### **2.3 Classificação das Técnicas de Verificação e Validação:**

Para garantir a Qualidade do Software, a abordagem das técnicas de Verificação e Validação de software podem ser organizadas em dois grandes grupos principais:

- **Técnicas Estáticas de Verificação e Validação de software;**
- **Técnicas Dinâmicas de Verificação e Validação de software;**

Ambas sem complementam e o ideal é que ambas abordagens sejam aplicadas na avaliação do produto.

### **2.4 Verificação e Validação de software por Técnicas Estáticas**

As Técnicas Estáticas são Inspeções e revisões que analisam os requisitos do sistema, modelos de projeto e o código-fonte do programa sem executá-lo. O objetivo dessas técnicas é identificar erros, inconsistências, ambiguidades e desvios de padrões e requisitos em um estágio inicial do ciclo de vida do desenvolvimento de software. Os 4 tipos de técnicas estáticas mais comuns são:

Table 2.3: Os tipos de TÉCNICAS ESTÁTICAS

- 
- a) Revisões Técnicas
  - b) Inspeções
  - c) Análise Estática
  - d) Verificação Formal e Métodos Formais
- 

#### 2.4.0.1 A) Revisões Técnicas

São atividades de controle de qualidade realizadas por engenheiros de software para descobrir erros na função, lógica ou implementação do software.

Podem ser **Informais** (sem a necessidade de agendamento ou declaração oficial ) ou **formais** ( com planilhas, documentação e acordos de compromisso).

#### 2.4.0.2 Passeio (Walkthrough) ( caso especial de revisão técnica formal)



Figure 2.1: Produtor repassando software com os revisores

Existe uma revisão técnica formal chamada “Passeio” onde o **produtor** “repassa” o artefato de software, explicando o material, enquanto os **revisores** levantam questões com base em sua preparação prévia.

#### 2.4.0.3 B) Inspeção do produto

Na Inspeção do Produto de software, uma **pequena equipe** verifica o código sistematicamente, procurando por possíveis erros e omissões. Tudo é executado e controlado minuciosamente com **planilhas** e **documentos de formalização** assinados pelos gestores das áreas.

## 2.4. VERIFICAÇÃO E VALIDAÇÃO DE SOFTWARE POR TÉCNICAS ESTÁTICAS 41



Figure 2.2: Equipe fazendo inspeção constantemente no software

### 2.4.0.4 C) Análise Estática

```
checker.py •
1
2
3 from http.client import HTTPConnection
4 from urllib.parse import urlparse
5
6 def site_is_online(url, timeout=2):
7     """Return True if the target URL is online,
8     Raise an exception otherwise."""
9     error = Exception("unknown error")
10    parser = urlparse(url)
11    host = parser.netloc or parser.path.split("/") [0]
12    for port in (80, 443):
13        connection = HTTPConnection(host=host, port=port,
14                                     timeout=timeout)
15        try:
16            connection.request("HEAD", "/")
```

Figure 2.3: Código python revisado “a olho”.

Inspeciona-se o código fonte do programa **sem executa-lo**, procurando erros de programação “na raça”.

### 2.4.0.5 D) Análise Estática Automatizada

Inspeciona-se o código fonte do programa **com auxílio de um programa**, como por exemplo o RATS (Rough Auditing Tool for Security) que significa “Ferramenta de auditoria bruta para segurança”.

```
C:\>Administrator: C:\Windows\system32\cmd.exe
C:\Users\Softpedia>C:\Users\Softpedia\Desktop\rats-2.3-win32\rats-2.3\rats.exe
Entries in perl database: 33
Entries in ruby database: 46
Entries in python database: 62
Entries in c database: 334
Entries in php database: 55
Total lines analyzed: 0
Total time 0.000000 seconds
2147483648 lines per second
C:\Users\Softpedia>
```

#### 2.4.0.6 E) Verificação Formal do Produto e Métodos Formais

Utiliza-se **métodos matemáticos e estatísticos** para avaliar o programa. Usado em software de missão crítica como software supervisório de usinas nucleares, cirurgia robótica e software de navegação de aviação.

## 2.5 Verificação e Validação de software por Técnicas Dinâmicas

As Técnicas Dinâmicas são **testes de software**, nos quais o sistema é executado com dados de testes simulados.

| Tipo de Teste           | Descrição   |
|-------------------------|---|
| <u>Teste de Unidade</u> | Concentra-se em <b>testar componentes individuais do software</b> , como módulos, classes ou funções, de forma isolada. O objetivo é verificar se cada unidade funciona corretamente em relação à sua especificação. Em um contexto orientado a objetos, isso inclui o teste de métodos dentro de uma classe. |

## 2.5. VERIFICAÇÃO E VALIDAÇÃO DE SOFTWARE POR TÉCNICAS DINÂMICAS43

| Tipo de Teste              | Descrição  |
|----------------------------|--|
| <u>Teste de Integração</u> | Após o teste de unidade, os <b>componentes são combinados e testados</b> em conjunto para <b>verificar as interações entre eles</b> . O teste de integração visa descobrir erros nas interfaces e na colaboração entre os módulos. |
| <u>Teste de Validação</u>  | Tem como objetivo <b>garantir que o software construído atende às expectativas e aos requisitos do cliente</b> . Os critérios de teste de validação são estabelecidos durante a análise de requisitos.                             |
| <u>Teste de Sistema</u>    | Testa o <b>software como um sistema completo</b> , após a integração de todos os componentes.  |

O teste de sistema pode ser separado em 6 subtestes:

|   |   |
|---|---|
| <u>Teste de Recuperação</u>                                     | Verifica a capacidade do sistema de se <b>recuperar de falhas</b> (software ou hardware) e continuar operando corretamente.   |
| <u>Teste de Segurança</u>                                       | Avalia se o sistema <b>protege dados e funcionalidades contra acessos não autorizados</b> e se cumpre os requisitos de privacidade e segurança.   |
| <u>Teste por Esforço</u>  | Examina o comportamento do sistema sob <b>condições de carga anormal</b> (volume de dados, número de usuários, etc.) para identificar seus limites e possíveis pontos de falha.   |
| <u>Teste de Desempenho</u>                                      | Avalia os <b>aspectos de desempenho do sistema</b> , como tempo de resposta, vazão e utilização de recursos, sob condições normais e de carga.  |
| <u>Teste de Disponibilização<br/>(Implantação/Configuração)</u> | Verifica se o software <b>opera corretamente em todos os ambientes</b> (plataformas, sistemas operacionais) para os quais foi projetado. Inclui também a avaliação dos procedimentos de instalação e da documentação associada. |

Teste de Regressão

É realizado após **alterações no software** (correção de erros, adição de novas funcionalidades) para garantir que as modificações não introduziram novos defeitos ou afetaram adversamente as partes existentes do sistema.

---

A escolha das técnicas dinâmicas e dos tipos de testes a serem utilizados depende do **tipo de software a ser desenvolvido**, dos **requisitos do projeto**, dos **recursos disponíveis** e dos **riscos envolvidos**. O objetivo final é **encontrar o maior número possível de erros** com o mínimo de esforço e garantir a entrega de software de alta qualidade que atenda às necessidades dos usuários.

## 2.6 Exercícios

### 2.6.1 Testes Sobre Verificação e Validação I

---

**TESTE 1**

Qual das seguintes afirmações melhor descreve o conceito de **verificação de software**?

- A) É o processo de testar o software no ambiente do usuário final para garantir que ele atenda às suas necessidades e expectativas.
  - B) Refere-se ao conjunto de atividades que visam descobrir erros e defeitos no software antes que ele seja entregue.
  - C) Consiste em garantir que o software construído implementa corretamente as funcionalidades e requisitos especificados.
  - D) Envolve a avaliação do desempenho do software em diferentes condições de carga e estresse para identificar gargalos.
  - E) É a prática de gerenciar e controlar as mudanças feitas no software ao longo do seu ciclo de vida.
- 

**TESTE 2**

Qual das seguintes afirmações melhor descreve o conceito de **validação de software**?

**TESTE 2**

---

- A) É o processo de confirmar se o software está livre de defeitos através da execução de diversos casos de teste que cobrem o código em sua totalidade.
  - B) Refere-se ao conjunto de atividades que garantem que o software foi construído corretamente, ou seja, em conformidade com as especificações de requisitos.
  - C) Envolve a análise estática do código-fonte para identificar potenciais vulnerabilidades de segurança e garantir a conformidade com padrões de codificação.
  - D) É a prática de documentar o design do software e garantir que a implementação esteja alinhada com a arquitetura definida.
  - E) Consiste em avaliar se o software atende às necessidades e expectativas do cliente e dos usuários finais, assegurando que o “produto certo” foi construído.
- 

**TESTE 3**

---

Qual das seguintes afirmações descreve melhor a técnica de “Walkthrough” (passeio) no contexto de Verificação e Validação de software?

- A) É uma técnica **dinâmica** que envolve a execução do software com dados de teste para observar seu comportamento e identificar defeitos em tempo de execução.
  - B) Refere-se à aplicação de **métodos formais** que utilizam notação matemática para especificar e verificar as propriedades do software.
  - C) Consiste na utilização de **testes automatizados** e ferramentas específicas para executar um grande número de casos de teste e verificar os resultados em relação às expectativas.
  - D) Trata-se de uma **revisão técnica informal** ou **passeio informal** realizado com colegas, onde um produto de software (como um documento de requisitos, um projeto ou um trecho de código) é examinado passo a passo por um grupo para identificar possíveis erros, inconsistências, ou áreas que precisam de melhorias, **sem a execução do software**.
  - E) Envolve a condução de **testes de aceitação pelo usuário** em um ambiente operacional para determinar se o software atende às necessidades do cliente e está pronto para ser implantado.
-

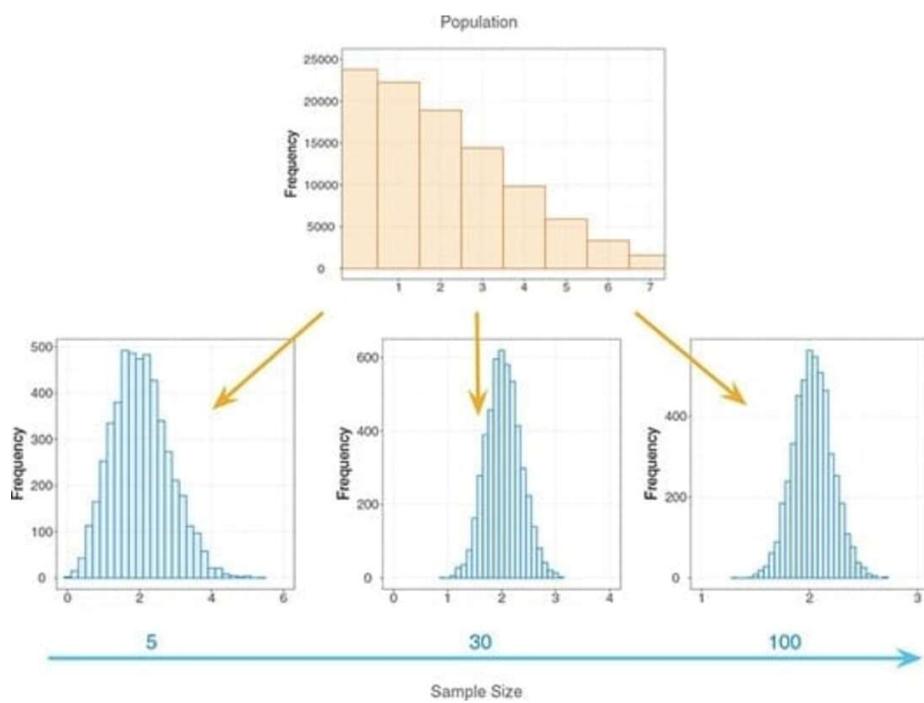


Figure 2.4: Um teste de hipótese para validar software

**TESTE 4**

Qual das seguintes opções lista os tipos mais comuns de **técnicas estáticas** utilizadas em Verificação e Validação de software?

- A) Revisões técnicas (incluindo inspeções e walkthroughs) e análise estática automatizada.
  - B) Testes de unidade, testes de integração e testes de sistema.
  - C) Testes alfa, testes beta e testes de aceitação pelo usuário.
  - D) Testes de desempenho, testes de segurança e testes de carga.
  - E) Depuração, teste de regressão e teste de fumaça.
- 

**TESTE 5**

Quem geralmente participa de uma inspeção formal de software (revisão técnica formal)?

- A) Apenas os desenvolvedores responsáveis pela criação do artefato inspecionado.
  - B) Uma equipe composta por diferentes papéis, como moderador, inspetor, relator e o autor do artefato.
  - C) Somente os gerentes de projeto para avaliar o progresso e a conformidade com o cronograma.
  - D) Exclusivamente os especialistas em testes para planejar os casos de teste futuros.
  - E) Apenas o cliente para garantir que os requisitos foram atendidos.
- 

**TESTE 6**

Qual a principal característica que distingue a verificação formal de outras técnicas de verificação e validação de software?

- A) A sua aplicação durante a fase de testes de unidade e integração.
  - B) O uso de métodos matemáticos e lógicos para provar a correção do software em relação às suas especificações.
  - C) A dependência da execução do software com dados de entrada reais.
  - D) O foco na identificação de defeitos de usabilidade na interface do usuário.
  - E) A sua realização por uma equipe de teste independente ao final do desenvolvimento.
-

**TESTE 7**

Qual o principal objetivo da aplicação de métodos formais na verificação de um produto de software?

- A) Melhorar a comunicação entre a equipe de desenvolvimento e os stakeholders.
  - B) Garantir que o software seja portável para diferentes plataformas de hardware.
  - C) Reduzir o tempo e o custo total do ciclo de vida do software.
  - D) Aumentar o nível de confiança na correção do software, através de provas matemáticas de suas propriedades.
  - E) Otimizar o desempenho do software em termos de velocidade e consumo de memória.
- 

**TESTE 8**

Qual das seguintes atividades é a principal característica das técnicas dinâmicas de verificação e validação de software?

- A) A análise estática do código-fonte em busca de possíveis defeitos.
  - B) A aplicação de métodos matemáticos para provar a correção do software.
  - C) A revisão manual da documentação do software para garantir a sua completude.
  - D) A execução do software com dados de entrada para observar seu comportamento e identificar erros.
  - E) A inspeção formal dos artefatos de software por uma equipe multidisciplinar.
- 

**TESTE 9**

Qual o principal objetivo do teste de software como uma técnica dinâmica de verificação e validação?

- A) Garantir que a documentação do software esteja completa e correta.
- B) Otimizar o desempenho do software em termos de velocidade e consumo de recursos.
- C) Encontrar erros no software, demonstrar que suas funções estão funcionando conforme as especificações e validar os requisitos.
- D) Verificar a conformidade do processo de desenvolvimento com os padrões estabelecidos.

---

TESTE 9

---

- E) Avaliar a usabilidade da interface do usuário do software.
- 

---

TESTE 10

---

Qual dos seguintes níveis de teste é considerado uma técnica dinâmica de verificação e validação que foca em exercitar a menor unidade testável do software?

- A) Teste de sistema.
  - B) Teste de integração.
  - C) Teste de validação.
  - D) Teste de unidade.
  - E) Teste de aceitação.
- 

### 2.6.2 Respostas para os Testes

| Testes   | Respostas |
|----------|-----------|
| Teste_01 | C         |
| Teste_02 | E         |
| Teste_03 | D         |
| Teste_04 | A         |
| Teste_05 | B         |
| Teste_06 | B         |
| Teste_07 | D         |
| Teste_08 | D         |
| Teste_09 | C         |
| Teste_10 | D         |

---



# Chapter 3

## Verificação de Validação de Software II - Continuação

### 3.1 Fundamentos de Teste de Software

**O objetivo primordial do teste de software é descobrir erros.** Um bom teste é aquele que tem alta probabilidade de encontrar um erro. Como benefício secundário, os testes demonstram que as funções do software estão funcionando de acordo com as especificações e que os requisitos relativos ao desempenho e ao comportamento parecem estar sendo atingidos. Os dados coletados durante os testes fornecem um bom indício da confiabilidade e da qualidade geral do software. No entanto, é fundamental lembrar que **os testes não podem mostrar a ausência de erros e defeitos, apenas que erros e defeitos de software estão presentes.** Por isso, não se deve considerar o teste como uma “rede de segurança” que detectará todos os erros decorrentes de práticas deficientes de engenharia de software.

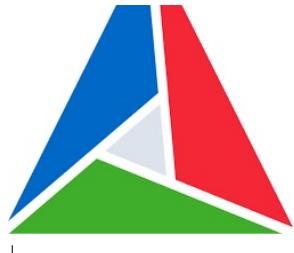
### 3.2 Os testes e o Ciclo de Vida do Software

#### 3.2.1 Teste de Unidade ou Unitários

Focado em testar cada componente individualmente para garantir que funcione adequadamente como uma unidade.

##### 3.2.1.0.1 Ferramentas (bibliotecas) de teste unitário das linguagens de programação mais populares e robustas

## 52CHAPTER 3. VERIFICAÇÃO DE VALIDAÇÃO DE SOFTWARE II - CONTINUAÇÃO

| Nome do framework de teste de unidade | Linguagem | Logotipo   |
|---------------------------------------|-----------|--|
| Pytest                                | Python    |   |
| JUnit                                 | Java      |   |
| CTest                                 | C++       |  |

### 3.2.1.1 Exemplo de Teste Unitário no Python

Vamos testar uma função chamada `soma` que faça adição de dois números `a` e `b`.

Crie um arquivo `soma.py` que contém função chamada `soma` que faça adição de dois números `a` e `b` :

Código do arquivo `soma` segue abaixo:

```
# arquivo soma.py

def soma(a, b):
    return a + b
```

Agora crie, no mesmo diretório, um arquivo chamado `test_soma.py` .

Esse arquivo cria a função de teste `test_soma_positivos()` que testa a função `soma` passando dois números 2 e 3 para ela.

O resultado esperado é 5 :

```
from soma import soma

def test_soma_positivos():
    assert soma(2, 3) == 5
```

Então, estando os arquivos **soma.py** e **test\_soma.py** no mesmo arquivo, basta executar a ferramenta **pytest**

```
===== test session starts =====
platform linux -- Python 3.x.x, pytest-x.x.x, py-x.x.x, pluggy-x.x.x
rootdir: /path/to/your/directory

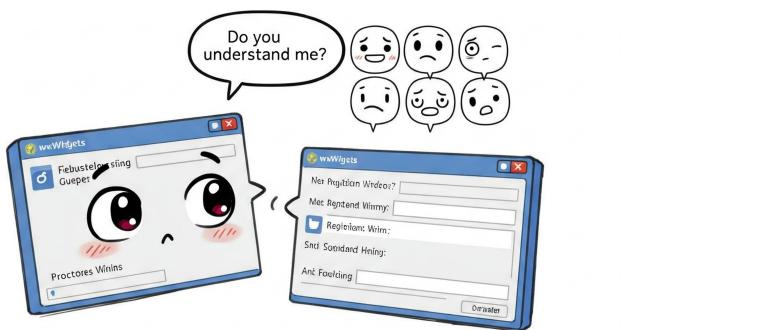
collected 5 items

test_soma.py ......

===== 5 passed in 0.01s =====
```

A função **soma()** passou no teste unitário.

### 3.2.2 Teste de Integração



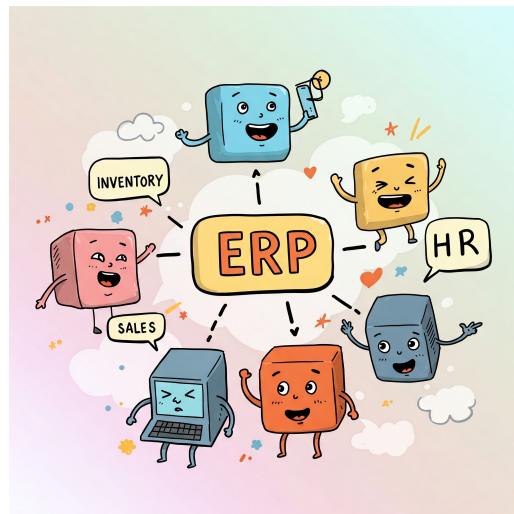
## 54 CHAPTER 3. VERIFICAÇÃO DE VALIDAÇÃO DE SOFTWARE II - CONTINUAÇÃO

Focado em testar cada componente individualmente para garantir que funcione adequadamente como uma unidade.

### 3.2.3 Teste de Validação

Focado em testar cada componente individualmente para garantir que funcione adequadamente como uma unidade.

### 3.2.4 Testes de Sistema



Focado em testar cada componente individualmente para garantir que funcione adequadamente como uma unidade.

### 3.3 Modelo V



Em Testes de Software, o **Diagrama V (ou Modelo V)** é uma variação na representação do modelo cascata (ciclo de vida clássico) que descreve a relação entre **ações de garantia da qualidade (testes)** e as **ações associadas a comunicação, modelagem e atividades de construção iniciais**. Ele oferece uma maneira de **visualizar como as ações de verificação e validação são aplicadas a um trabalho de engenharia anterior**.

Em outras palavras, o modelo V correlaciona os **testes de Verificação e Validação de Software** ao **ciclo de Vida** do processo de desenvolvimento de Software fornecendo a noção que o software é testado em todo seu **ciclo de vida**:

| Etapa do Ciclo de Vida do Processo de Desenvolvimento de Software   | Qualidade Testes de Verificação e Validação do Software  |
|---|--|
| <ul style="list-style-type: none"><li>• Elicitação de Requisitos</li><li>• Planejamento: Arquitetura do sistema</li><li>• Planejamento: Arquitetura dos Módulos</li><li>• Codificação</li></ul> | <ul style="list-style-type: none"><li>• Teste de aceitação</li><li>• Teste de Sistema</li><li>• Teste de Integração</li><li>• Teste Unitário</li></ul> |

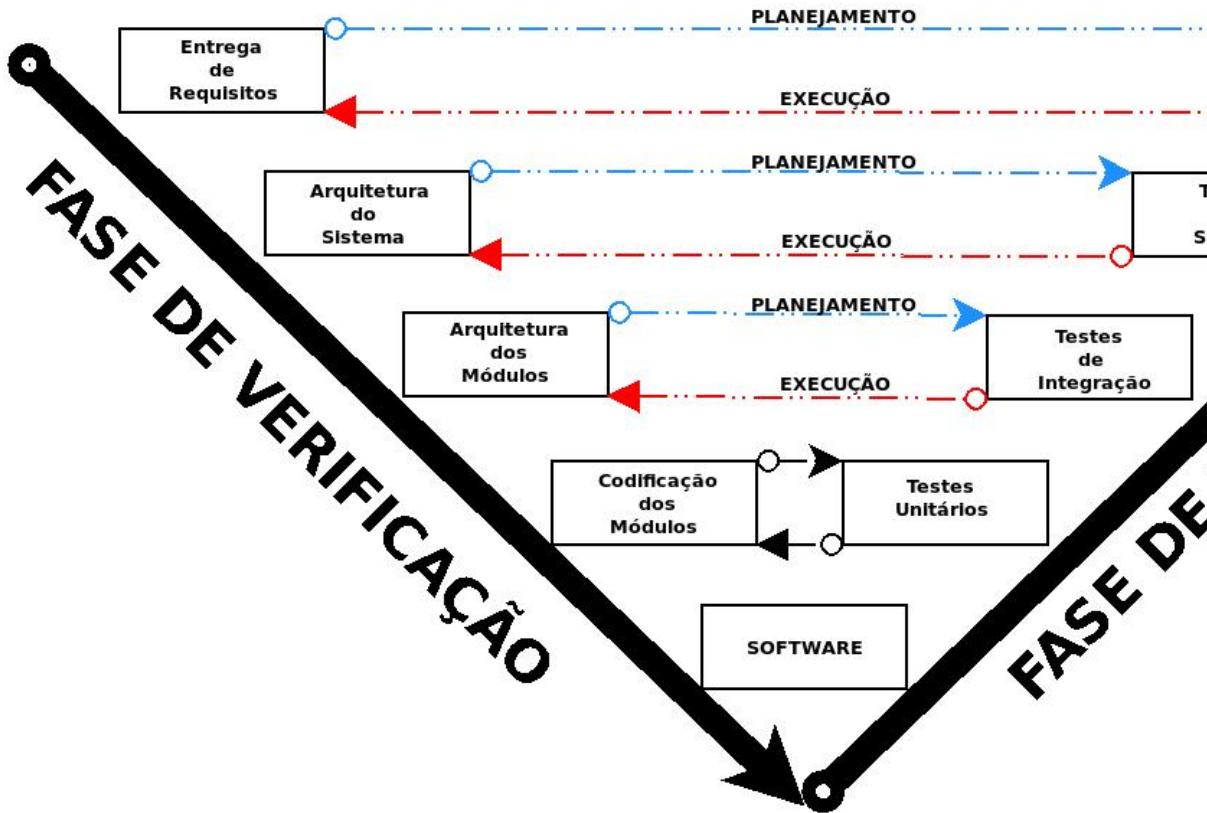
---

#### SOFTWARE PRONTO

---

Esse correlacionamento pode ser visualizado na figura abaixo, em formato “V”:

## MODELO "V"



### 3.4 Testes Unitários

O objetivo primordial do teste unitário é **focar o esforço de verificação na menor unidade de projeto do software para descobrir erros dentro dos limites dessa unidade**. Ele busca garantir que cada parte individual do sistema funcione corretamente.

O teste unitário se concentra na **lógica interna de processamento e nas estruturas de dados dentro dos limites de um componente**. Ele examina os caminhos de controle importantes para descobrir erros na lógica do módulo.

### 3.4.1 Testes Estruturais (Caixa-Branca)



Os testes **caixa-branca**, também chamados de teste da caixa-de-vidro ou **teste estrutural**, são uma filosofia de projeto de casos de teste que utiliza a estrutura de controle descrita como parte do projeto no nível de componentes para derivar casos de teste.

O teste de caixa-branca se alinha com o teste unitário. O teste caixa-branca é frequentemente aplicado a **pequenos componentes de programas (por exemplo, módulos ou pequenos grupos de módulos)**, sendo considerado um “teste no pequeno”.

### 3.4.2 Testes Funcionais (Caixa-Preta)



Os **testes caixa-preta**, também chamados de **teste comportamental** ou **teste funcional**, são uma abordagem de teste que **focaliza os requisitos funcionais do software**. Diferentemente dos testes caixa-branca, que examinam a estrutura lógica interna do software, o teste caixa-preta **faz referência a testes realizados na interface do software**, com **pouca preocupação em relação à estrutura lógica interna do software**. Em vez de olhar o código-fonte, os testadores caixa-preta trabalham com a **visão externa do software**.

#### 3.4.2.1 Técnicas de testes funcionais

## 3.5 Exercícios

### 3.5.1 Testes

---

TESTE 01

---

Qual é o objetivo fundamental do Teste de Software ?

- a) Otimizar o desempenho do software para garantir uma melhor experiência do usuário.
  - b) Descobrir o maior número possível de erros e defeitos no software antes de sua entrega ou implantação.
  - c) Validar se o software está sendo desenvolvido dentro do prazo e do orçamento estipulados.
  - d) Garantir que o software seja compatível com todas as plataformas e dispositivos existentes.
  - e) Documentar detalhadamente todas as funcionalidades do software para referência futura.
- 

---

TESTE 02

---

No contexto de Testes de Software, qual a distinção essencial entre **verificação** e **validação**?

- a) Verificação foca em testar o software em diferentes ambientes, enquanto validação se concentra na sua funcionalidade.
  - b) Verificação é realizada pelos desenvolvedores, e validação é feita pelos usuários finais.
  - c) Verificação pergunta “Estamos construindo o produto corretamente?”, enquanto validação pergunta “Estamos construindo o produto certo?”
  - d) Validação ocorre antes da verificação no ciclo de vida de desenvolvimento de software.
  - e) Ambas se referem ao mesmo conjunto de atividades de garantia da qualidade.
- 

---

TESTE 03

---

Qual tipo de teste de software se baseia no **exame da estrutura interna** do software, incluindo seu código-fonte, para projetar casos de teste?

- a) Teste de caixa-preta
- b) Teste de desempenho.
- c) Teste de usabilidade.
- d) Teste de caixa-branca (ou estrutural)

## 60CHAPTER 3. VERIFICAÇÃO DE VALIDAÇÃO DE SOFTWARE II - CONTINUAÇÃO

---

### TESTE 03

---

- e) Teste de integração.
- 

---

### TESTE 4

---

Por que é importante realizar testes em diferentes níveis (unidade, integração, validação, sistema)?

- A) Para reduzir o custo total do processo de teste.
  - B) Para facilitar a comunicação entre as equipes de desenvolvimento e teste.
  - C) Porque diferentes tipos de erros são mais facilmente detectados em diferentes níveis de teste.
  - D) Para garantir que todos os membros da equipe participem do processo de teste.
  - E) Para cumprir as exigências de normas e padrões de qualidade como a ISO 9000.
- 

---

### TESTE 5

---

Qual é a principal característica que o **Modelo V** busca explicitar em relação ao ciclo de vida clássico?

- A) A natureza iterativa e incremental do desenvolvimento.
  - B) A forte ênfase na comunicação com o cliente em todas as fases.
  - C) O uso extensivo de prototipagem para validação precoce dos requisitos.
  - D) A priorização da flexibilidade e adaptabilidade a mudanças nos requisitos.
  - E) A relação entre as fases de desenvolvimento iniciais e as ações de garantia da qualidade (testes).
- 

---

### TESTE 6

---

**Modelo V**, à medida que a equipe de software “desce” pelo lado esquerdo do “V”, qual é o foco principal das atividades?

- A) Implementação e codificação do sistema.
- B) Execução de testes unitários e de integração.
- C) Refinamento dos requisitos básicos em representações cada vez mais detalhadas do problema e da solução.

---

TESTE 6

---

- D) Implantação e suporte contínuo do software.
  - E) Gerenciamento do projeto e controle das mudanças.
- 

---

TESTE 7

---

No **Modelo V**, qual tipo de atividade de garantia da qualidade está tipicamente associada à fase de especificação de requisitos?

- A) Teste de unidade.
  - B) Teste de integração.
  - C) Teste de sistema.
  - D) Teste de aceitação (validação dos requisitos).
  - E) Teste de desempenho.
- 

---

TESTE 8

---

Segundo o **Modelo V**, as atividades de teste no lado direito do “V” têm como objetivo principal:

- A) Otimizar o desempenho do software.
  - B) Validar cada um dos modelos criados à medida que a equipe “desceu” pelo lado esquerdo.
  - C) Garantir a segurança do sistema contra ameaças.
  - D) Documentar o código-fonte de forma detalhada.
  - E) Facilitar a manutenção futura do software.
- 

---

TESTE 9

---

No **Modelo V**, o teste de sistema, que está ligado à fase de projeto da arquitetura do sistema, visa:

- A) Testar as interações entre os componentes.
- B) Avaliar a facilidade de manutenção do código.
- C) Validar os requisitos do cliente em um ambiente de produção.
- D) Garantir que cada unidade de código funcione corretamente.

TESTE 9

---

- E) Verificar se o software integrado funciona conforme o especificado nos requisitos do sistema.
- 

TESTE 10

---

Qual a principal vantagem de visualizar o processo de teste através do **Modelo V**?

- A) Facilita a adoção de metodologias ágeis.
  - B) Reduz a necessidade de documentação detalhada.
  - C) Oferece uma maneira clara de como as ações de verificação e validação se relacionam com as atividades de desenvolvimento.
  - D) Garante a automação completa de todos os testes.
  - E) Elimina a necessidade de revisões técnicas.
- 

### 3.5.2 Respostas dos Testes

| Teste    | Resposta |
|----------|----------|
| Teste_01 | B        |
| Teste_02 | C        |
| Teste_03 | D        |
| Teste_04 | C        |
| Teste_05 | E        |
| Teste_06 | C        |
| Teste_07 | D        |
| Teste_08 | B        |
| Teste_09 | E        |
| Teste_10 | C        |

### 3.5.3 Questões Dissertativas:

---

**Questão 1**

Discuta a **importância da distinção entre verificação e validação** no contexto de testes de software. Explique como cada uma dessas atividades contribui para a garantia da qualidade do produto final, citando as definições apresentadas nas fontes.

---

**Resposta:**

---

**Questão 2**

Compare e contraste as abordagens de teste de **caixa-branca** e **caixa-preta**. Para cada abordagem, descreva seus focos principais, as informações necessárias para sua aplicação. Avalie as vantagens e desvantagens de cada uma na detecção de diferentes tipos de defeitos.

---

**Resposta:**

---

**Questão 3**

Explore a **relação entre os diferentes níveis de teste** (unidade, integração, validação e sistema) no ciclo de vida do software. Explique os objetivos específicos de cada nível e como eles se complementam para garantir a qualidade em diferentes granularidades do sistema.

---

**Resposta:**

### 3.5.4 Respostas Questões Dissertativas:

---

**Resposta questão 1**

A distinção entre **verificação** e **validação** é fundamental em testes de software para garantir a qualidade sob diferentes perspectivas.

**Verificação** busca responder à pergunta: “**Estamos criando o produto corretamente?**”. Envolve um conjunto de tarefas que asseguram que o software foi construído e pode ser rastreado segundo os requisitos do cliente. Isso inclui diversas atividades de garantia da qualidade de software (SQA), como revisões técnicas, auditorias de qualidade e configuração, monitoramento de desempenho e revisão de documentação.

Por outro lado, a **validação** procura responder: “**Estamos criando o produto certo?**”. Ela se refere a um conjunto de tarefas que asseguram que o software criado atende às reais necessidades dos usuários. A validação de software é alcançada por meio de uma série de testes que demonstram conformidade com os requisitos.

Um plano de teste descreve as classes de testes a serem realizados para garantir que todos os requisitos funcionais, características comportamentais, conteúdo, requisitos de desempenho e documentação estejam corretos.

Ambas as atividades são cruciais: a verificação garante que o software está sendo construído de acordo com as especificações, enquanto a validação assegura que as especificações atendem às necessidades do cliente, contribuindo assim para um produto final de alta qualidade.

---

### **Resposta questão 2**

As abordagens de teste de software podem ser amplamente categorizadas em **caixa-branca** (ou *white-box*) e **caixa-preta** (ou *black-box*). O **teste caixa-branca** foca na **estrutura interna do software**, incluindo sua lógica, código e fluxo de controle. Para aplicar testes de caixa-branca, é necessário **conhecimento detalhado do código-fonte** e do design do software. A vantagem do teste caixa-branca é sua capacidade de **descobrir erros lógicos internos**, caminhos não percorridos e problemas na implementação. No entanto, pode ser **difícil e demorado** para sistemas complexos e não detecta necessariamente erros relacionados aos requisitos ou à usabilidade.

Em contraste, o **teste caixa-preta** examina a **funcionalidade do software** do ponto de vista do usuário, **sem conhecimento de sua implementação interna**. As informações necessárias são os **requisitos do software**, as especificações e a interface do usuário. A principal vantagem do teste caixa-preta é sua capacidade de **validar os requisitos do software** e identificar problemas na funcionalidade e usabilidade percebidas pelo usuário. Sua desvantagem é que ele **não garante a cobertura de toda a lógica interna** do software e pode não detectar certos tipos de erros de implementação.

Ambas as abordagens são importantes e complementares para garantir uma qualidade abrangente do software.

---

### **Resposta questão 3**

A estratégia de teste de software geralmente envolve múltiplos níveis que abordam a qualidade em diferentes granularidades do sistema.

O **teste de unidade** é o primeiro nível e se concentra em **testar individualmente cada unidade** (componente, classe ou função) do software, conforme implementado no código-fonte. O objetivo é **verificar a funcionalidade básica e a correção** de cada parte isoladamente.

Após as unidades serem testadas, o próximo nível é o **teste de integração**, que se concentra em **testar as interfaces e a interação entre os diferentes componentes** que foram integrados para construir a arquitetura do software. O objetivo é **descobrir erros nas interfaces entre módulos** e os efeitos colaterais causados pela adição de novas unidades.

O **teste de sistema** avalia o **software integrado como um todo**, juntamente com outros elementos do sistema (hardware, outros softwares, pessoas). O objetivo é **aprovar o software quando ele é incorporado em um sistema maior** e verificar os requisitos do sistema. Tipos de teste de sistema incluem teste de recuperação, teste de segurança, teste de esforço e teste de desempenho. Esses níveis de teste são complementares e progressivos. Finalmente, o **teste de aceitação (validação)** ocorre após a integração e tem como objetivo **demonstrar a conformidade do software com os requisitos estabelecidos** como parte da modelagem de requisitos. Ele busca responder à pergunta se o software construído atende às necessidades do cliente. Os critérios de validação, derivados dos requisitos do software, formam a base para essa fase de teste.

Resumindo: o teste de unidade ajuda a identificar e corrigir erros nas partes menores do software, o teste de integração garante que essas partes funcionem juntas corretamente, o teste de sistema assegura que o software funcione de forma adequada dentro do sistema completo e o teste de aceitação (validação) verifica se o software atende aos requisitos do cliente. Cada nível foca em diferentes tipos de defeitos, contribuindo para uma garantia de qualidade mais abrangente.

---

#### Resposta questão 4

---

---

#### Resposta questão 5

---

---

#### Resposta questão 6

---

---

**Resposta questão 7**

---

---

**Resposta questão 8**

---

---

**Resposta questão 9**

---

---

**Resposta questão 10**

---

---

---

### 3.6 Cadastro de Clientes

Acesso ao Banco de Dados na núvem POSTGRES para você testar o seu:

---

|          |  |
|----------|--|
| host:    | pg-ads-engs2-miguel7penteadoads-engs2.c.aivencloud.com |
| porta:   | 17135  |
| usuario: | SEU RA   |
| senha:   | SEU RA   |
| banco:   | banco-dados-ra   |
| SSL:     | require  |

---

OBS: substitua “ra” pelo seu ra, obviamente.

Cliente para testar via celular:

Android Postgresql Client

[https://play.google.com/store/apps/details?id=rafrobsystems.postgresclient&pcampaignid=web\\_share](https://play.google.com/store/apps/details?id=rafrobsystems.postgresclient&pcampaignid=web_share)



### 3.6.1 Tabela Clientes

```
CREATE TABLE clientes
(
    id      varchar(15) unique not null,
    nome    varchar(500) not null ,
    endereco  varchar(500) not null ,
    nascimento date
);
```

## 3.7 Cadastro de Fornecedores

## 3.8 Cadastro de Produtos





## Chapter 4

# Introdução à Manutenção de Software



## 4.1 1- Manutenção: definição e características”

### 4.1.1 Definição de Manutenção de Software:



*“Manutenção de software é um processo contínuo de mudança em um sistema após sua liberação para uso.” - Roger Pressman - Engenharia de Software 8a edição*

Segundo Pressman, a Manutenção de Software envolve diversas ações:

1. **Correção de erros:** Identificação e reparo de defeitos encontrados no software;
2. **Adaptação:** Modificação do software para que ele continue funcionando em um ambiente operacional ou de negócios em mudança. Isso pode incluir adaptações a novas plataformas, sistemas operacionais ou requisitos de hardware;
3. **Melhorias:** Implementação de novas características e funcionalidades solicitadas pelos clientes ou outras partes interessadas;
4. **Suporte ao usuário:** Fornecimento de assistência aos usuários para resolver dúvidas sobre instalação, operação e uso da aplicação;

O processo de modificação de um produto de software após a entrega, para corrigir defeitos, melhorar o desempenho ou outros atributos [Sommerville]. A manutenção é uma parte importante da evolução do software [Sommerville].

### 4.1.2 Natureza da Mudança:

Discutir por que o software precisa ser mantido, incluindo correções de bugs, adaptação a novos ambientes, implementação de novos requisitos e manutenção preventiva [Sommerville, Pressman].

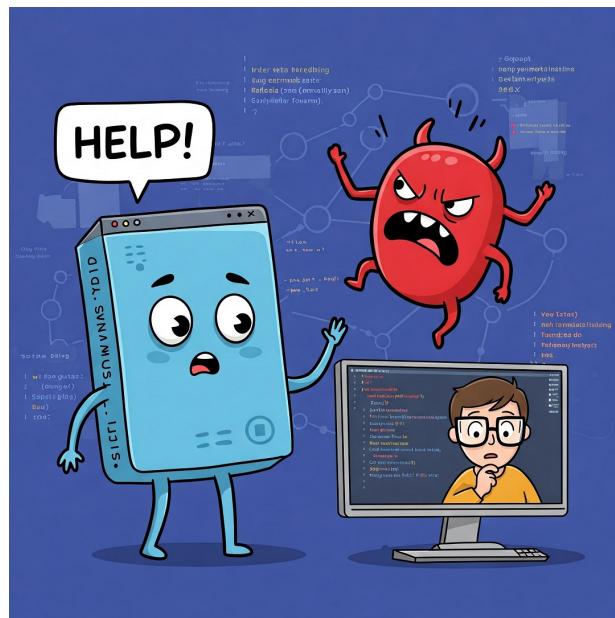
A Primeira Lei da Engenharia de Sistemas afirma que *não importa em qual estágio do ciclo de vida, o sistema mudará* [Sommerville - referindo-se à inevitabilidade da mudança].

### 4.1.3 Tipos de Manutenção:

**Padrão ISO/IEC 14764 - Engenharia de Software — Processos de Ciclo de Vida — Manutenção**

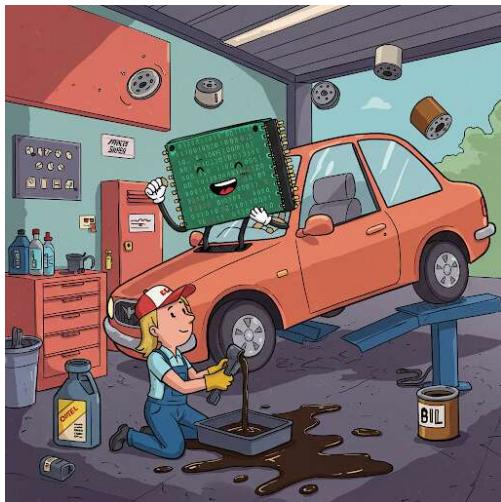
#### 4.1.3.1 MANUTENÇÕES VOLTADAS A CORREÇÃO DO SOFTWARE

##### 4.1.3.2 Manutenção Corretiva



É uma manutenção de REAÇÃO a um defeito encontrado para repará-lo.

#### 4.1.3.3 Manutenção Preventiva



É uma manutenção de Pró-Ação para melhorar a capacidade de manutenibilidade do sistema. Por exemplo, um software específico com arquivo de configuração construído em linguagem XML pode ser reescrito em formato JSON.

#### 4.1.3.4 VOLTADAS AO APRIMORAMENTO DO SOFTWARE (ENHACEMENT)

#### 4.1.3.5 Manutenção Adaptativa



É uma manutenção de REAÇÃO a uma mudança de ambiente. Exemplo, frente a um novo Navegador (javascript), o cliente web precisa ter seus métodos reescritos.

#### 4.1.3.6 Manutenção Perfectiva (FUNCIONAL ou EVOLUTIVA)



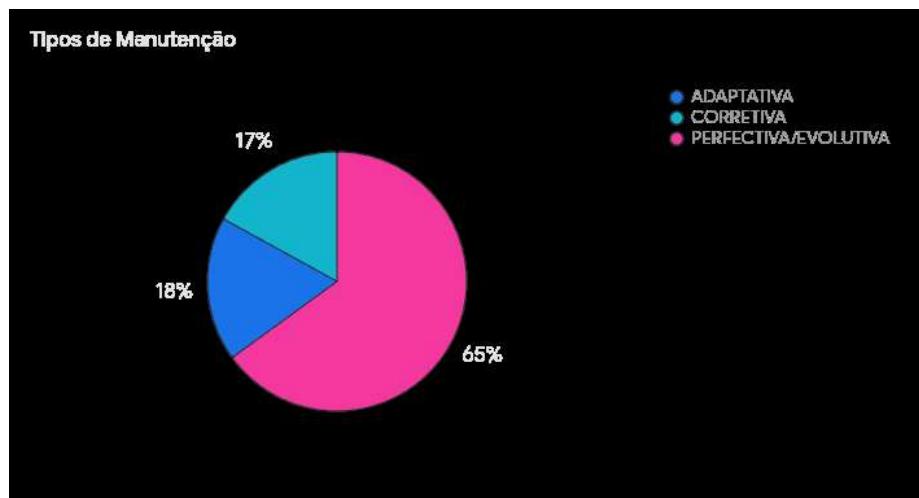
É uma manutenção de Pró-Ação para adicionar FUNCIONALIDADE ou DESEMPENHO ao software ou Sistema de Informação.

#### 4.1.3.7 Distribuição de tempo por tipo de manutenção (Sommerville, 2008)

Adaptativa: 18%

Corretiva: 17%

Perfectiva/Evolutiva: 65%



#### 4.1.4 Custos da Manutenção:

Mencionar que os custos de manutenção podem frequentemente exceder os custos iniciais de desenvolvimento [Sommerville, Pressman].

### 4.2 2- Introdução à Manutenibilidade

#### 4.2.1 Definição Preliminar:

Apresentar o conceito de manutenibilidade como a facilidade com que o software pode ser modificado [Sommerville, Pressman]. A manutenibilidade é um atributo essencial de um bom software [Sommerville] e um indicativo qualitativo da facilidade de corrigir, adaptar ou melhorar o software [Pressman, 74].

#### 4.2.2 Importância da Manutenibilidade:

Discutir por que a manutenibilidade é crucial para reduzir os custos e o esforço da manutenção a longo prazo [Sommerville, Pressman]. Qualidade e facilidade de manutenção são resultantes de um projeto bem feito [Pressman, 24].

### 4.3 Exercícios

#### 4.3.1 Fatoração I

Baixe o código SQL da tabela Clientes, do arquivo clientes1.sql da pasta exercicios/fatoracao. Acrescente chave primária a tabela clientes, definindo o atributo cpf como a chave primária. Faça a publicação do código refatorado.

```
git clone https://github.com/miguel7penteado/ADS-EngenhariaSoftware2025.git
```

Teste o seu código no seu servidor postgres da núvem.

Resposta:

Testei o código a seguir no servidor para adicionar CPF como chave primaria da tabela clientes:

```
alter table public.clientes add constraint "chave_primaria_clientes" primary key(cpf);
```

O código final do arquivo clientes1.sql ficaria assim, depois de refatorado:

```
create table public.clientes
(
    cpf integer not null,
    nome varchar(200),
    idade integer,
    endereco varchar(500),
    nascimento varchar(100)
);

alter table public.clientes add constraint "chave_primaria_clientes" primary key(cpf);
```

# **Chapter 5**

## **APROFUNDANDO A MANUTENIBILIDADE E AS TÉCNICAS DE DESENVOLVIMENTO**

### **5.1 Manutenibilidade**

#### **5.1.1 Atributos da Manutenibilidade:**

Detalhar as características que influenciam a manutenibilidade, como modularidade, clareza do código, documentação adequada, complexidade, acoplamento e coesão [Pressman, Sommerville]. A boa prática de engenharia de software demanda modularidade efetiva, com alta coesão e baixo acoplamento [Pressman, 45].

#### **5.1.2 Métricas de Manutenibilidade:**

Introduzir a ideia de que a manutenibilidade pode ser avaliada e até mesmo medida através de métricas de software [Pressman, Sommerville]. O livro “Engenharia-de-software-8-ed-roger-pressman.pdf” menciona “Métricas para manutenção” no Capítulo 30.

## 5.2 Técnicas de Desenvolvimento para a Manutenibilidade

### 5.2.1 Princípios de Projeto:

Discutir como princípios de projeto como separação de interesses, abstração, encapsulamento e baixo acoplamento impactam positivamente a manutenibilidade [Pressman, Sommerville]. A separação de interesses é um princípio-chave de projeto e implementação de software [Sommerville, 127].

### 5.2.2 Qualidade do Código:

Enfatizar a importância de práticas de codificação limpa, convenções de estilo consistentes e evitar construções complexas para facilitar a compreensão e modificação do código.

### 5.2.3 Documentação:

Ressaltar a necessidade de documentação interna (comentários no código) e externa (manuais, diagramas de projeto) para auxiliar na manutenção [Sommerville, Pressman].

## **Chapter 6**

# **PROCESSOS E PADRÕES NA MANUTENÇÃO DE SOFTWARE**

(Making off da aula )

### **6.1 Processos de Manutenção**

#### **6.1.1 Fluxo do Processo de Manutenção**

Apresentar as etapas típicas envolvidas em um processo de manutenção, como identificação da necessidade de mudança, análise da solicitação, projeto da modificação, implementação, teste e implantação [Pressman, Sommerville].

#### **6.1.2 Gerenciamento de Mudanças**

Discutir a importância de um processo formal de gerenciamento de mudanças para controlar as alterações aplicadas ao software durante a manutenção [Pressman, Sommerville]. O Princípio 6 da prática da engenharia de software é “Gerencie mudanças” [Pressman, 44]. O gerenciamento de configuração (abordado em nossa aula anterior) está intimamente ligado ao gerenciamento de mudanças na manutenção [Pressman, Sommerville].

## 6.2 Padrões de Desenvolvimento

### 6.2.1 Impacto na Manutenção

Explicar como a adoção de padrões de desenvolvimento bem estabelecidos (arquiteturais, de projeto, de implementação) pode melhorar significativamente a manutibilidade, promovendo consistência e compreensão do código [Pressman, Sommerville]. O projeto baseado em padrões é considerado [Pressman, 5, 13, 354].

### 6.2.2 Exemplos de Padrões Relevantes

Apresentar exemplos de padrões que favorecem a manutenibilidade, como padrões de projeto GoF (Observer [Sommerville, 107]), padrões arquiteturais (camadas [Pressman, 15]), etc.

## 6.3 Padrões de Manutenção

### 6.3.1 Conceito e Exemplos

Introduzir a ideia de padrões específicos para atividades de manutenção, como padrões de refatoração [Pressman, 12, 52] para melhorar a estrutura do código sem alterar seu comportamento externo.

## **Chapter 7**

# **ABORDAGENS MODERNAS E ATIVIDADES DE APOIO À MANUTENÇÃO**

(Making Off)

## **7.1 Desenvolvimento Baseado em Componentes e Impactos na Manutenção**

### **7.1.1 Conceito de Desenvolvimento Baseado em Componentes (DBC):**

Apresentar o DBC como uma abordagem que enfatiza a construção de sistemas a partir de componentes de software reutilizáveis [Pressman, Sommerville]. O desenvolvimento baseado em componentes é um modelo de processo especializado [Pressman, 9, 52].

### **7.1.2 Impactos na Manutenção:**

Discutir como o DBC pode influenciar a manutenção, facilitando a substituição, atualização ou reutilização de componentes, mas também introduzindo desafios relacionados à compatibilidade e dependências [Pressman].

## 7.2 Desenvolvimento Orientado a Aspectos e Impactos na Manutenção

### 7.2.1 Conceito de Desenvolvimento Orientado a Aspectos (DOA):

Introduzir o DOA como uma técnica para modularizar interesses transversais (aspectos) que podem estar espalhados por vários módulos em um sistema orientado a objetos tradicional [Pressman, Sommerville]. O desenvolvimento de software orientado a aspectos é um modelo de processo especializado [Pressman, 9, 54].

### 7.2.2 Impactos na Manutenção:

Explicar como o DOA pode melhorar a manutibilidade ao isolar e gerenciar esses interesses transversais, tornando o código mais limpo e facilitando modificações em funcionalidades como logging, segurança ou tratamento de erros [Pressman, Sommerville].

## 7.3 Atividades de Apoio a Manutenção

### 7.3.1 Gerenciamento de Configuração na Manutenção:

Reforçar a importância do GC (que discutimos em uma aula anterior) para rastrear e controlar as mudanças realizadas durante a manutenção [Pressman, Sommerville].

### 7.3.2 Reengenharia:

Introduzir o conceito de reengenharia como uma forma de melhorar a manutibilidade de sistemas legados através da reestruturação ou reimplementação [Pressman, Sommerville]. O Capítulo 36 do livro “Engenharia-de-software-8-ed-roger-pressman.pdf” trata de “Manutenção e reengenharia”.

### 7.3.3 Testes de Regressão:

Destacar a necessidade de testes de regressão para garantir que as modificações realizadas durante a manutenção não introduzam novos defeitos [Pressman, Sommerville].

## Chapter 8

# Gerência de Configuração



(aula em processo de edição)

## 8.1 Introdução à Gerência de Configuração

### 8.1.1 Definição de Gerência de Configuração (GC):

GC é o nome do processo geral de gerenciamento de um sistema de software em mudança. O objetivo do gerenciamento de configuração é apoiar o processo de integração do sistema para que todos os desenvolvedores possam acessar o código do projeto e os documentos relacionados de forma controlada, descobrir quais mudanças foram feitas, bem como compilar e ligar componentes para criar um sistema.

### 8.1.2 A Natureza da Mudança em Software:

A mudança é uma realidade para grandes sistemas. As necessidades e requisitos organizacionais se alteram durante a vida útil de um sistema, bugs precisam ser reparados e os sistemas necessitam se adaptar às mudanças em seu ambiente. De fato, a Primeira Lei da Engenharia de Sistemas afirma que não importa em qual estágio do ciclo de vida, o sistema mudará.

### 8.1.3 Importância da GC:

Sem o gerenciamento de configuração, as mudanças aplicadas ao sistema podem ocorrer de forma descontrolada, levando a inconsistências, perda de trabalho e dificuldades na manutenção e evolução do software. A GC garante que as mudanças sejam aplicadas ao sistema de uma forma controlada.

## 8.2 Elementos da Gerência de Configuração

### 8.2.1 Itens de Configuração de Software (ICIs)

Os itens que compõem todas as informações produzidas como parte do processo de software são chamados coletivamente de configuração de software. Isso inclui programas de computador (código fonte e executável), produtos que descrevem os programas (documentação para diversos stakeholders) e dados ou conteúdo. Um ICI é um elemento de informação com nome, que pode variar desde um simples diagrama UML até um documento de projeto completo. Diferentes versões de um ICI podem existir.

### 8.2.2 Identificação:

Cada ICI deve ter um nome único para permitir seu rastreamento e gerenciamento.

### 8.2.3 Controle de Versão:

Suporte para manter o controle das diferentes versões de ICIs ao longo do tempo. Isso permite rastrear o histórico de mudanças, reverter para versões anteriores e gerenciar múltiplas linhas de desenvolvimento.

#### **8.2.4 Controle de Mudanças:**

O processo de garantia de que as mudanças em sistemas e componentes sejam registradas e mantidas para que as mudanças sejam gerenciadas e todas as versões de componentes sejam identificadas e armazenadas por todo o tempo de vida do sistema. Isso geralmente envolve solicitação de mudança, avaliação, aprovação e implementação controlada das alterações. Um formulário de solicitação de mudança pode ser utilizado.

#### **8.2.5 Auditoria de Configuração:**

Avaliações para garantir que os ICIs e seus registros correspondam à configuração real do software em um determinado momento.

#### **8.2.6 Relatório de Status:**

Documentação e comunicação sobre o status dos ICIs e das mudanças realizadas.

### **8.3 Processo de Gerência de Configuração**

#### **8.3.1 Planejamento da GC:**

Definição das políticas, procedimentos e ferramentas a serem utilizadas para a GC.

| <b>Etapa</b>                                   | <b>Descrição</b>  |
|--|---|
| <b>1-Identificação da Configuração</b>         | Seleção dos itens de trabalho que serão controlados pela GC.  |
| <b>2-Controle de Mudanças</b>                  | Implementação do processo para gerenciar solicitações de mudança, incluindo:<br>Solicitação formal de alteração.<br>Avaliação da alteração (impacto, custo, etc.).<br>Aprovação da alteração (geralmente por um Grupo de Controle de Alterações).<br>Implementação da alteração.<br>Verificação da alteração. |
| <b>3-Liberação da Configuração</b>             | Preparação e disponibilização de versões específicas do software para teste, implantação ou entrega.  |
| <b>4-Auditoria e Relatório da Configuração</b> | Verificação da conformidade com o plano de GC e comunicação do status da configuração.  |

## 8.4 Ferramentas de Gerência de Configuração

Muitas ferramentas de gerenciamento de configurações foram desenvolvidas para dar suporte aos processos de GC. Elas variam desde ferramentas simples que oferecem suporte a uma única tarefa (como rastreamento de bugs) até conjuntos complexos e caros de ferramentas integradas que oferecem suporte a todas as atividades de GC. Exemplos de funcionalidades comuns em ferramentas de GC incluem:

- \* Armazenamento e gerenciamento de versões de arquivos.
- \* Controle de acesso e permissões.
- \* Rastreamento de mudanças e histórico.
- \* Suporte a ramificações (branches) e merges.
- \* Gerenciamento de solicitações de mudança.
- \* Construção automatizada de sistemas.
- \* Ambientes de Desenvolvimento Colaborativo (CDEs) como GForge, OneDesk e Rational Team

## 8.5 GC em Contextos Ágeis e Tradicionais

A necessidade de gerenciamento de configuração é fundamental para todos os grandes sistemas desenvolvidos por equipes. Métodos ágeis também desenvolveram suas próprias terminologias de GC, às vezes para distinguir a abordagem ágil dos métodos tradicionais. Mesmo em desenvolvimento ágil, onde a mudança é bem-vinda, a GC é essencial para manter a organização e o controle sobre o software em evolução.

## 8.6 Referências:

Leitura dos Capítulos 25 de “Engenharia-de-software-9-ed-Ian-Sommerville.pdf”

Leitura dos Capítulos 29 de “Engenharia-de-software-8-ed-roger-pressman.pdf”.

\* Explorar ferramentas de Gerência de Configuração como Git, SVN, etc.

## Chapter 9

# Revisão para NP2



## Chapter 10

# Revisão para a Substitutiva



## Chapter 11

# Referencias



# Chapter 12

## Apêndice I - Estudo da - ERP Agrotec

### 12.1 Entrega #01 - Módulo Cadastros

### 12.2 Interface JanelaPrincipal

Os arquivos estão na pasta ProjetoERP-AGROTE\01ModuloCadastros\03codificacao\source do repositório da disciplina

Arquivo *ERPAgroTech.py*

```
# -*- coding: utf-8 -*-

#####
## Python code generated with wxFormBuilder (version 4.2.1-0-g80c4cb6)
## http://www.wxformbuilder.org/
##
## PLEASE DO *NOT* EDIT THIS FILE!
#####

import wx
import wx.xrc

import gettext
_ = gettext.gettext
```

```
from CadastroClientes import TipoCadastroClientes

#####
## Class TipoJanelaPrincipal
#####

class TipoJanelaPrincipal ( wx.Frame ):

    def __init__( self, parent ):
        wx.Frame.__init__ ( self, parent, id = wx.ID_ANY, title = _(u"ERP AGROTEC - Ent...") )

        self.SetSizeHints( wx.DefaultSize, wx.DefaultSize )

        self.TipoMenuPrincipal = wx.MenuBar( 0 )
        self.TipoMenuArquivo = wx.Menu()
        self.TipoMenuItemSair = wx.MenuItem( self.TipoMenuArquivo, wx.ID_ANY, _(u"Sair") )
        self.TipoMenuArquivo.Append( self.TipoMenuItemSair )

        self.TipoMenuPrincipal.Append( self.TipoMenuArquivo, _(u"Arquivo") )

        self.TipoMenuCadastro = wx.Menu()
        self.TipoMenuItemClientes = wx.MenuItem( self.TipoMenuCadastro, wx.ID_ANY, _(u"Cadast...") )
        self.TipoMenuCadastro.Append( self.TipoMenuItemClientes )

        self.TipoMenuItemFornecedores = wx.MenuItem( self.TipoMenuCadastro, wx.ID_ANY, _(u"Fornece...") )
        self.TipoMenuCadastro.Append( self.TipoMenuItemFornecedores )

        self.TipoMenuItemProdutos = wx.MenuItem( self.TipoMenuCadastro, wx.ID_ANY, _(u"Produtos") )
        self.TipoMenuCadastro.Append( self.TipoMenuItemProdutos )

        self.TipoSubmenuRelatorios = wx.Menu()
        self.TipoMenuItemRelatorioClientes = wx.MenuItem( self.TipoSubmenuRelatorios, wx.ID_ANY, ... )
        self.TipoSubmenuRelatorios.Append( self.TipoMenuItemRelatorioClientes )

        self.TipoMenuItemRelatorioFornecedores = wx.MenuItem( self.TipoSubmenuRelatorios, wx.ID_ANY, ... )
        self.TipoSubmenuRelatorios.Append( self.TipoMenuItemRelatorioFornecedores )

        self.TipoMenuItemRelatorioProdutos = wx.MenuItem( self.TipoSubmenuRelatorios, wx.ID_ANY, ... )
        self.TipoSubmenuRelatorios.Append( self.TipoMenuItemRelatorioProdutos )

        self.TipoMenuCadastro.AppendSubMenu( self.TipoSubmenuRelatorios, _(u"Relatórios") )

        self.TipoMenuPrincipal.Append( self.TipoMenuCadastro, _(u"Cadastro") )

        self.SetMenuBar( self.TipoMenuPrincipal )
```

```
self.TipoBarraStatus = self.CreateStatusBar( 1, wx.STB_SIZEGRIP, wx.ID_ANY )
self.TipoBarraStatus.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_INFOBK ) )

self.Centre( wx.BOTH )

# Connect Events
self.Bind( wx.EVT_MENU, self.EventoTerminarPrograma,
           id = self.TipoMenuItem)
self.Bind( wx.EVT_MENU, self.EventoAbrePainelClientes,
           id = self.TipoMenuItem)
self.Bind( wx.EVT_MENU, self.EventoAbrePainelFornecedores,
           id = self.TipoMenuItem)
self.Bind( wx.EVT_MENU, self.EventoAbrePainelProdutos,
           id = self.TipoMenuItem)
self.Bind( wx.EVT_MENU, self.EventoAbrePainelRelatorioClientes,
           id = self.TipoMenuItem)
self.Bind( wx.EVT_MENU, self.EventoAbrePainelRelatorioFornecedores,
           id = self.TipoMenuItem)
self.Bind( wx.EVT_MENU, self.EventoAbrePainelRelatorioProdutos,
           id = self.TipoMenuItem)

def __del__( self ):
    pass

def MakeModal(self, modal=True):
    if modal and not hasattr(self, '_disabler'):
        self._disabler = wx.WindowDisabler(self)
    if not modal and hasattr(self, '_disabler'):
        del self._disabler

# Virtual event handlers, override them in your derived class
def EventoTerminarPrograma( self, event ):
    event.Skip()

def EventoAbrePainelClientes( self, event ):
    janelaClientes = TipoCadastroClientes(None)
    janelaClientes.MakeModal()
    janelaClientes.Show()

def EventoAbrePainelFornecedores( self, event ):
    event.Skip()

def EventoAbrePainelProdutos( self, event ):
    event.Skip()

def EventoAbrePainelRelatorioClientes( self, event ):
    event.Skip()

def EventoAbrePainelRelatorioFornecedores( self, event ):
    event.Skip()
```

```
def EventoAbrePainelRelatorioProdutos( self, event ):
    event.Skip()
```

Arquivo main.py

```
# -*- coding: utf-8 -*-

import wx

from ERPAgroTech import TipoJanelaPrincipal

class Programa(TipoJanelaPrincipal):
    def __init__(self, parent):
        TipoJanelaPrincipal.__init__(self, parent)

    def OnCloseWindow(self, event):
        self.Close(True)
        event.Skip()

app = wx.App(False)      # cria uma nova aplicação e não redireciona stdout e stderr para a janela
frame = Programa(None)  # frame é uma janela de nível de topo
frame.MakeModal()
frame.Show()             # Mostra a janela
app.MainLoop()           # aplicação entra em loop até finalizar
```

### 12.2.1 Como executar a janela principal

1. Baixar o e instalar o Python (preferencialmente a versão 3.9 para Windows 10 ou 11)
2. Abrir uma janela do MS-DOS (prompt de comando) e mandar o utilitário **PIP** instalar o pacote **wxpython**:

```
pip install --upgrade wxpython
```

3. Abrir uma janela do MS-DOS (prompt de comando) e mandar o utilitário e baixar o repositório da disciplina com a ferramenta GIT:

```
git clone git@github.com:miguel7penteado/ADS-EngenhariaSoftware2025.git
```

4. Pelo MS-DOS entrar na pasta **ProjetoERP-AGROTEC\01ModuloCadastros\03codificacao\source** :

```
cd ADS-EngenhariaSoftware2025\ProjetoERP-AGROTEC\01ModuloCadastros\03codificacao\source
```

5. Pelo MS-DOS mandar o interpretador python executar o ERP AGROTEC

```
python3 main.py
```

## 12.3 Cadastro de Clientes

Acesso ao Banco de Dados na nûvem POSTGRES para você testar o seu:

---

|          |  |
|----------|--|
| host:    | pg-ads-engs2-miguel7penteadoads-engs2.c.aivencloud.com |
| porta:   | 17135  |
| usuario: | SEU RA   |
| senha:   | SEU RA   |
| banco:   | banco-dados-ra   |
| SSL:     | require  |

---

OBS: substitua “ra” pelo seu ra, obviamente.

Cliente para testar via celular:

Android Postgresql Client

[https://play.google.com/store/apps/details?id=rafrobsystems.postgresclient&pcampaignid=web\\_share](https://play.google.com/store/apps/details?id=rafrobsystems.postgresclient&pcampaignid=web_share)



### 12.3.1 Tabela Clientes

```
CREATE TABLE clientes
(
    id      varchar(15) unique not null,
    nome   varchar(500) not null ,
    endereco  varchar(500) not null ,
    nascimento date
);
```

## 12.4 Cadastro de Fornecedores

## 12.5 Cadastro de Produtos

## Chapter 13

## Apendice II



# Chapter 14

# Apendice III