



Instituto Federal de Educação, Ciência e Tecnologia da Bahia - IFBA  
Departamento de Ciência da Computação  
Tecnólogo em Análise e Desenvolvimento de Sistemas

## Testes de software

André L. R. Madureira <[andre.madureira@ifba.edu.br](mailto:andre.madureira@ifba.edu.br)>  
Doutorando em Ciência da Computação (UFBA)  
Mestre em Ciência da Computação (UFBA)  
Engenheiro da Computação (UFBA)

# Requisitos de software

---

- Descrições do que o sistema deve fazer, os serviços que ele oferece e as restrições a seu funcionamento
- Os requisitos refletem as necessidades dos clientes
- Podem ser classificados de acordo com seu **grau de detalhamento**:
  - **Requisitos de usuário**: descrição abstrata de alto nível, usando linguagem natural com diagramas
  - **Requisitos do sistema**: descrição detalhada dos serviços e restrições do sistema, definindo exatamente o que deve ser implementado.

# Requisitos de software

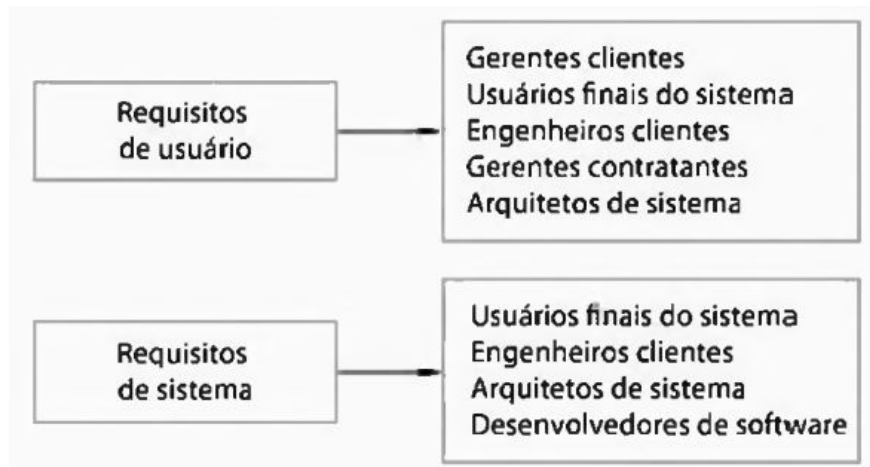
Os requisitos de sistema podem ser organizados em um documento (**especificação funcional**).

- Descrições do que o sistema deve fazer, os serviços que ele oferece e as restrições a seu funcionamento
- Os requisitos refletem as necessidades dos clientes
- Podem ser classificados de acordo com seu **grau de detalhamento**:
  - **Requisitos de usuário**: descrição abstrata de alto nível, usando linguagem natural com diagramas
  - **Requisitos do sistema**: descrição detalhada dos serviços e restrições do sistema, definindo exatamente o que deve ser implementado.

# Requisitos de software

- Porque ter requisitos com diferentes níveis de detalhamento?
  - Diferentes pessoas têm diferentes necessidades de compreensão sobre um sistema

**Ex:** um gerente ou usuário de um sistema bancário estão mais preocupados com os serviços fornecidos pelo sistema, do que em como um sistema vai ser implementado



# Exemplo de Requisitos de software

## **Requisitos de usuário:**

1. O sistema deve gerar relatórios gerenciais mensais que mostrem o custo dos medicamentos prescritos por cada clínica durante aquele mês.

## **Requisitos de sistema:**

- 1.1 No último dia útil de cada mês deve ser gerado um resumo dos medicamentos prescritos, seus custos e as prescrições de cada clínica.
- 1.2 Após 17:30h do último dia útil do mês, o sistema deve gerar automaticamente o relatório para impressão.
- 1.3 Um relatório será criado para cada clínica, listando os nomes dos medicamentos, o número total de prescrições, o número de doses prescritas e o custo total dos medicamentos prescritos.

# Classificação de Requisitos de software

---

## **Requisitos funcionais (RF):**

Descreve o comportamento do sistema perante determinadas entradas, bem como os serviços que ele deve fornecer

## **Requisitos não-funcionais (RNF):**

São restrições impostas sobre os serviços e funções oferecidos pelo sistema.

# Classificação de Requisitos de software

---

## Requisitos funcionais (RF):

Descreve o comportamento do sistema perante determinadas entradas, bem como os serviços que ele deve fornecer

- **RF1:** Sacar dinheiro no caixa
  - **RF2:** Emitir extrato
  - **RF3:** Alterar senha
- **RF4:** Solicitar empréstimo
  - **RF5:** Investir dinheiro

## Requisitos não-funcionais (RNF):

São restrições impostas sobre os serviços e funções oferecidos pelo sistema.

- **RNF1:** Sacar da conta-corrente só se saldo  $> 0$
- **RNF2:** Alterar senha só se nova senha possuir 6 dígitos
- **RNF3:** Tempo máximo para ficar com App do banco aberto

# Testes de software

---

- **Objetivos:**
  - Mostrar ao cliente e ao desenvolvedor que um software se adequa aos seus requisitos (funcionais e não-funcionais)
  - Descobrir condições que levam ao software se comportar de maneira incorreta / indesejável ou diferente das especificações



# Testes de software

---

- **Objetivos:**

- Mostrar ao cliente e ao desenvolvedor que um software se adequa aos seus requisitos (funcionais e não-funcionais)
- Descobrir condições que levam ao software se comportar de maneira incorreta / indesejável ou diferente das especificações

Os testes não podem demonstrar se o software é livre de defeitos ou se ele se comportará conforme especificado em qualquer situação.

*“Os testes podem **mostrar apenas a presença de erros**, e não a sua ausência.”*  
Edsger Dijkstra

# Testes de Software Manuais x Automatizados

---

- **Testes manuais**

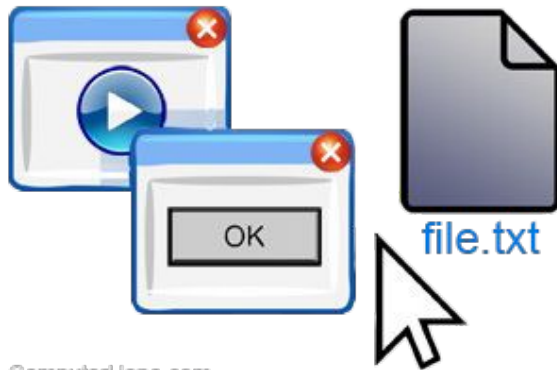
- Um testador executa o programa com alguns dados de teste e compara os resultados com suas expectativas

- **Testes automatizados**

- Os testes são codificados em um programa que é executado cada vez que o sistema em desenvolvimento é testado
- Os testes nunca poderão ser totalmente automatizados, pois é difícil testar sistemas que possuem estado de forma automatizada (*como testar efeitos colaterais ?*)

# Como testar automaticamente uma interface gráfica (GUI)?

---



- Construimos um teste por tela?
- Como saber se o comportamento da interface esta correto?
- Tiramos *printscreens* da tela do computador?

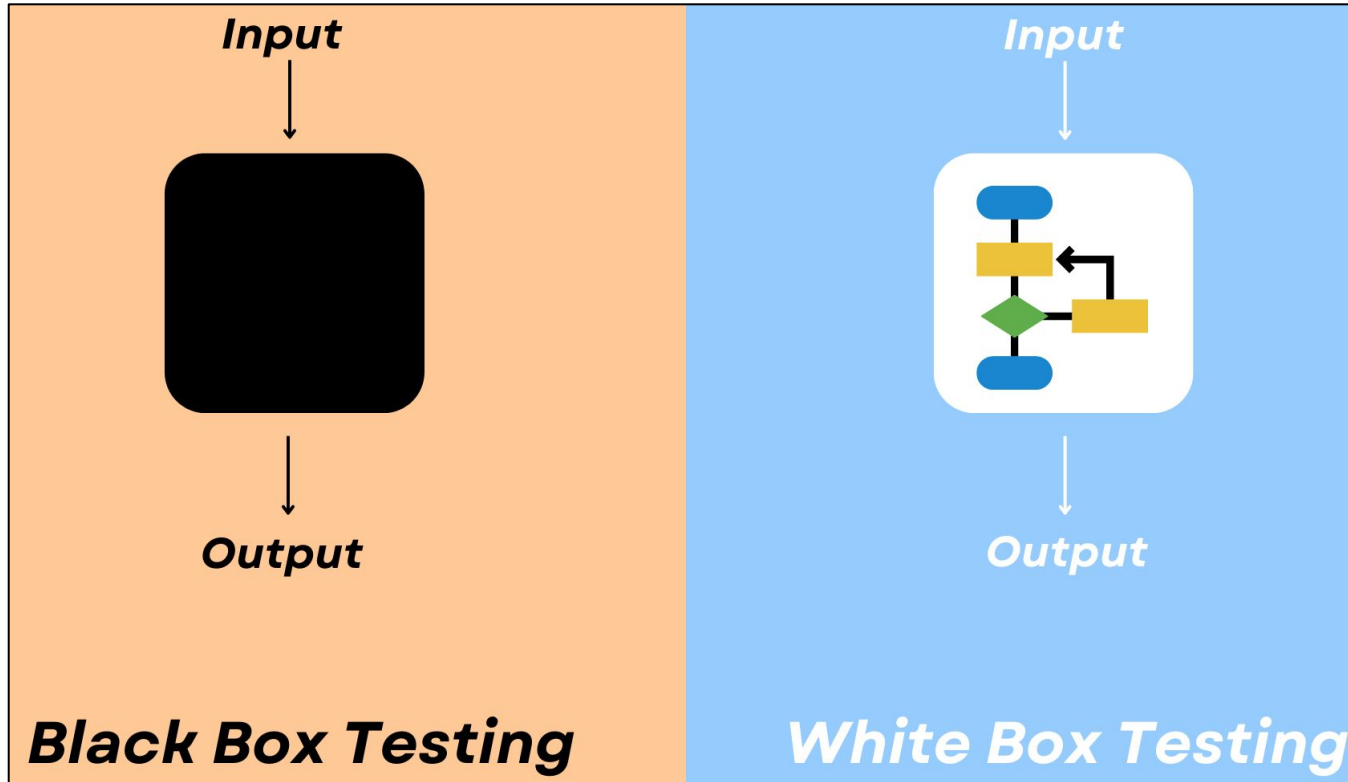
- Como imitamos a movimentação e cliques de mouse de uma pessoa?
- Temos que simular o teclado também?
- Como a velocidade de uso do teclado e mouse afeta a interface?

# Testes de Software Caixa Preta x Branca

---

- Ambos os testes que demonstram que o sistema funciona conforme esperado
- **Testes de caixa preta**
  - Não se conhece a lógica ou implementação interna do sistema
  - Teste de funcionalidade esperada do sistema
- **Testes de caixa branca**
  - Se conhece a lógica e implementação do sistema
  - Teste dos caminhos de execução interna do sistema

# Testes de Software Caixa Preta x Branca



# Testes de Software Caixa Preta x Branca

---

- **Testes de caixa branca são superiores aos de caixa preta?**
  - Testes de caixa branca podem garantir que um sistema não apresenta falhas, desde que:
    - Todos os caminhos de execução sejam testados
  - **Problema:** A quantidade de caminhos de execução é muito grande
    - Testar todos os caminhos de execução é inviável, mesmo para softwares pequenos

# Exercício

Considerando as atividades de especificação, verificação e validação de software, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Os requisitos descrevem o que um sistema deve fazer, os serviços que ele oferece e as restrições a seu funcionamento. Eles estão organizados em um documento chamado de relatório de requisitos.

II - Os requisitos podem ser classificados em requisitos de usuário e de sistema. Os requisitos de usuário descrevem a implementação dos serviços e restrições do sistema. Já os requisitos do sistema são uma descrição abstrata do software.

III - Outra forma de classificar os requisitos é através do agrupamento em requisitos funcionais ou não-funcionais. Os requisitos funcionais descrevem o comportamento do sistema, enquanto que os não-funcionais trazem restrições impostas sobre serviços e funções. Como exemplos de requisitos funcionais e não-funcionais temos "Emitir relatório somente no final do mês" e "Emitir extrato", respectivamente.

IV - Os testes de software tem por objetivo mostrar ao cliente e ao desenvolvedor que o sistema está em conformidade com seus requisitos. Desta forma, os testes de software garantem a ausência de erros no sistema.

- ☐ Somente I, II e IV.
- ☐ Somente II e IV.
- ☐ Somente I e II.
- ☐ Somente III e IV.
- ☐ Nenhuma das alternativas anteriores.

# Exercício

Considerando as atividades de especificação, verificação e validação de software, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Os requisitos descrevem o que um sistema deve fazer, os serviços que ele oferece e as restrições a seu funcionamento. Eles estão organizados em um documento chamado de **relatório de requisitos**. **F** [Slide 3: Requisitos de software](#)

II - Os requisitos podem ser classificados em requisitos de usuário e de sistema. Os requisitos de usuário descrevem a implementação dos serviços e restrições do sistema. Já os requisitos do sistema são uma descrição abstrata do software.

III - Outra forma de classificar os requisitos é através do agrupamento em requisitos funcionais ou não-funcionais. Os requisitos funcionais descrevem o comportamento do sistema, enquanto que os não-funcionais trazem restrições impostas sobre serviços e funções. Como exemplos de requisitos funcionais e não-funcionais temos "Emitir relatório somente no final do mês" e "Emitir extrato", respectivamente.

IV - Os testes de software tem por objetivo mostrar ao cliente e ao desenvolvedor que o sistema está em conformidade com seus requisitos. Desta forma, os testes de software garantem a ausência de erros no sistema.

- ☐ Somente I, II e IV.
- ☐ Somente II e IV.
- ☐ Somente I e II.
- ☐ Somente III e IV.
- ☐ Nenhuma das alternativas anteriores.



# Exercício

Considerando as atividades de especificação, verificação e validação de software, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Os requisitos descrevem o que um sistema deve fazer, os serviços que ele oferece e as restrições a seu funcionamento. Eles estão organizados em um documento chamado de relatório de requisitos. **F**

II - Os requisitos podem ser classificados em requisitos de usuário e de sistema. Os requisitos de usuário descrevem a implementação dos serviços e restrições do sistema. Já os requisitos do sistema são uma descrição abstrata do software. **F**

III - Outra forma de classificar os requisitos é através do agrupamento em requisitos funcionais ou não-funcionais. Os requisitos funcionais descrevem o comportamento do sistema, enquanto que os não-funcionais trazem restrições impostas sobre serviços e funções. Como exemplos de requisitos funcionais e não-funcionais temos "Emitir relatório somente no final do mês" e "Emitir extrato", respectivamente.

IV - Os testes de software tem por objetivo mostrar ao cliente e ao desenvolvedor que o sistema está em conformidade com seus requisitos. Desta forma, os testes de software garantem a ausência de erros no sistema.

- ☐ Somente I, II e IV.
- ☐ Somente II e IV.
- ☐ Somente I e II.
- ☐ Somente III e IV.
- ☐ Nenhuma das alternativas anteriores.

[Slide 3: Requisitos de software](#)

# Exercício

Considerando as atividades de especificação, verificação e validação de software, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Os requisitos descrevem o que um sistema deve fazer, os serviços que ele oferece e as restrições a seu funcionamento. Eles estão organizados em um documento chamado de relatório de requisitos. **F**

II - Os requisitos podem ser classificados em requisitos de usuário e de sistema. Os requisitos de usuário descrevem a implementação dos serviços e restrições do sistema. Já os requisitos do sistema são uma descrição abstrata do software. **F**

III - Outra forma de classificar os requisitos é através do agrupamento em requisitos funcionais ou não-funcionais. Os requisitos funcionais descrevem o comportamento do sistema, enquanto que os não-funcionais trazem restrições impostas sobre serviços e funções. Como exemplos de requisitos funcionais e não-funcionais temos "Emitir relatório somente no final do mês" e "Emitir extrato" respectivamente. **F**

IV - Os testes de software tem por objetivo mostrar ao cliente e ao desenvolvedor que o sistema está em conformidade com seus requisitos. Desta forma, os testes de software garantem a ausência de erros no sistema.

- ☐ Somente I, II e IV.
- ☐ Somente II e IV.
- ☐ Somente I e II.
- ☐ Somente III e IV.
- ☐ Nenhuma das alternativas anteriores.

[Slide 7: Classificação de Requisitos de software](#)

# Exercício

Considerando as atividades de especificação, verificação e validação de software, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Os requisitos descrevem o que um sistema deve fazer, os serviços que ele oferece e as restrições a seu funcionamento. Eles estão organizados em um documento chamado de relatório de requisitos. **F**

II - Os requisitos podem ser classificados em requisitos de usuário e de sistema. Os requisitos de usuário descrevem a implementação dos serviços e restrições do sistema. Já os requisitos do sistema são uma descrição abstrata do software. **F**

III - Outra forma de classificar os requisitos é através do agrupamento em requisitos funcionais ou não-funcionais. Os requisitos funcionais descrevem o comportamento do sistema, enquanto que os não-funcionais trazem restrições impostas sobre serviços e funções. Como exemplos de requisitos funcionais e não-funcionais temos "Emitir relatório somente no final do mês" e "Emitir extrato" respectivamente. **F**

IV - Os testes de software tem por objetivo mostrar ao cliente e ao desenvolvedor que o sistema está em conformidade com seus requisitos. Desta forma, os testes de software garantem a ausência de erros no sistema. **F**

[Slide 9: Testes de software](#)

- ☐ Somente I, II e IV.
- ☐ Somente II e IV.
- ☐ Somente I e II.
- ☐ Somente III e IV.
- ☒ Nenhuma das alternativas anteriores.

# Casos de teste

---

- Casos de teste são utilizados para testar o sistema ou parte dele
- Os casos de teste são um **conjunto de entradas** (dados de teste) e **saídas esperadas** do sistema (os resultados do teste)

# Casos de teste

---

- Casos de teste são utilizados para testar o sistema ou parte dele
- Os casos de teste são um **conjunto de entradas** (dados de teste) e **saídas esperadas** do sistema (os resultados do teste)
  - É impossível criar casos de teste automaticamente
    - *“Somente o desenvolvedor sabe o comportamento esperado do sistema que ele deseja testar”* (SOMMERVILLE, 2003)
  - Porém os dados dos testes e a execução deles podem ser automatizados

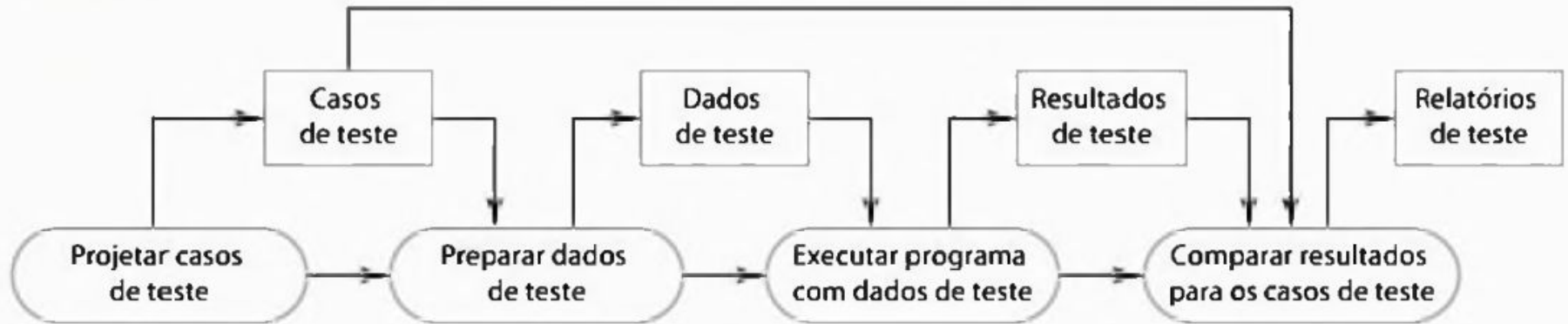
# Testes em diferentes abordagens de desenvolvimento de software

---

- **Processo de software dirigido a planos**
  - São elaborados planos de testes a partir das especificações e do projeto do sistema. Os planos são executados por uma equipe de testadores independentes.
- **Extreme programming**
  - Testes são executados em paralelo com o levantamento de requisitos, antes de se iniciar o desenvolvimento
- **Desenvolvimento incremental (ou desenvolvimento dirigido a testes)**
  - Testes executados para cada incremento desenvolvido

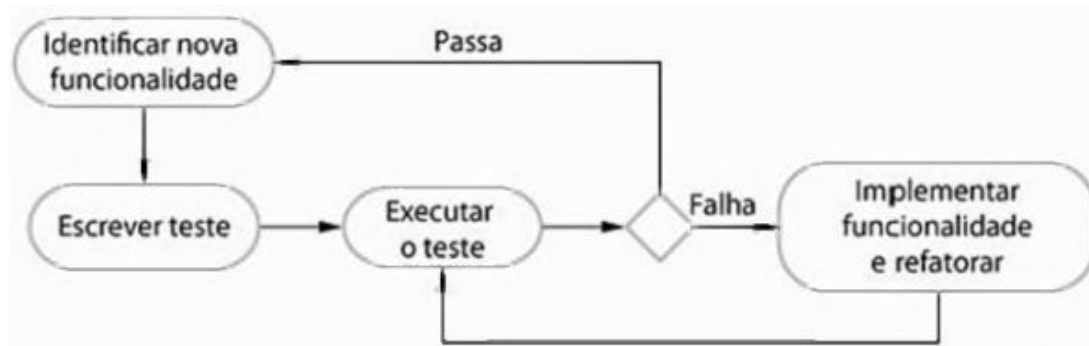
# Processo de software dirigido a planos

---



# Desenvolvimento incremental (dirigido a testes)

- Desenvolvimento e testes incrementais, executados de forma intercalada
  - Somente passa para o próximo incremento, se o incremento atual passar em todos os testes
- **Etapas:**





# Desenvolvimento incremental (dirigido a testes)

---

- **Vantagens:**

**Cobertura de código:**

Cada incremento (segmento de código) precisa ser testado. Ajuda na compreensão do código pelos desenvolvedores.

# Desenvolvimento incremental (dirigido a testes)

---

- **Vantagens:**

**Cobertura de código:**

Cada incremento (segmento de código) precisa ser testado. Ajuda na compreensão do código pelos desenvolvedores.

**Teste de regressão:**

Testes que permitem verificar se novos bugs não foram introduzidos no sistema, conforme este foi desenvolvido

# Desenvolvimento incremental (dirigido a testes)

---

- **Vantagens:**

**Cobertura de código:**

Cada incremento (segmento de código) precisa ser testado. Ajuda na compreensão do código pelos desenvolvedores.

**Teste de regressão:**

Testes que permitem verificar se novos bugs não foram introduzidos no sistema, conforme este foi desenvolvido

**Depuração simplificada:**

Quando um teste falha, a localização do problema deve ser óbvia. Logo, não é necessário o uso de ferramentas de depuração para localizar o erro.

# Desenvolvimento incremental (dirigido a testes)

---

- **Vantagens:**

**Cobertura de código:**

Cada incremento (segmento de código) precisa ser testado. Ajuda na compreensão do código pelos desenvolvedores.

**Teste de regressão:**

Testes que permitem verificar se novos bugs não foram introduzidos no sistema, conforme este foi desenvolvido

**Depuração simplificada:**

Quando um teste falha, a localização do problema deve ser óbvia. Logo, não é necessário o uso de ferramentas de depuração para localizar o erro.

**Documentação de sistema facilitada:**

Os testes em si mesmos agem como uma forma de documentação que descreve o que o código deve estar fazendo

# Classificação de Testes de software

---

- Testes de software podem ser classificados de acordo com os seus objetivos:
  - **Testes de validação**
    - O sistema é testado usando casos de teste, que refletem o uso esperado do sistema
  - **Testes de defeitos**
    - O sistema é testado usando casos de testes projetados para expor defeitos (comportamentos incorretos, indesejáveis, inesperados, ou anomalias), sem se preocupar com o uso esperado do sistema

# Verificação e Validação de software (V & V)

---

- Sistemas não devem ser testados como uma unidade única
  - O sistema deve ser testado por partes, usando testes isolados para que, no fim, essas partes sejam integradas e testadas em conjunto

# Verificação e Validação de software (V & V)

---

- Sistemas não devem ser testados como uma unidade única
  - O sistema deve ser testado por partes, usando testes isolados para que, no fim, essas partes sejam integradas e testadas em conjunto

Quando um defeito é descoberto em um dos estágios, o programa precisa ser corrigido (**depurado**)

# Verificação e Validação de software (V & V)

---

- Sistemas não devem ser testados como uma unidade única
  - O sistema deve ser testado por partes, usando testes isolados para que, no fim, essas partes sejam integradas e testadas em conjunto

Quando um defeito é descoberto em um dos estágios, o programa precisa ser corrigido (**depurado**)

Após a depuração, alguns testes precisam ser executados novamente



# Verificação e Validação de software (V & V)

---

- Sistemas não devem ser testados como uma unidade única
  - O sistema deve ser testado por partes, usando testes isolados para que, no fim, essas partes sejam integradas e testadas em conjunto

Quando um defeito é descoberto em um dos estágios, o programa precisa ser corrigido (**depurado**)

Após a depuração, alguns testes precisam ser executados novamente

Por isso o processo de testes é um conjunto de etapas iterativas

# Verificação e Validação de software (V & V)

---

- Os testes são parte de um amplo processo de verificação e validação de software (V&V)
- **Verificação:** atestar que um software atende a seus **requisitos** funcionais e não-funcionais
- **Validação:** garantir que o software atende as **expectativas** do cliente

# Verificação e Validação de software (V & V)

---

- Os testes são parte de um amplo processo de verificação e validação de software (V&V)
- **Verificação:** atestar que um software atende a seus **requisitos** funcionais e não-funcionais
- **Validação:** garantir que o software atende as **expectativas** do cliente
  - As expectativas do cliente nem sempre são traduzidas de maneira correta em requisitos (i.e., podem estar faltando requisitos, ou as suas descrições podem estar incompletas)

# Verificação e Validação de software (V & V)

---

- Os testes são parte de um amplo processo de verificação e validação de software (V&V)
- **Verificação:** atestar que um software atende a seus **requisitos** funcionais e não-funcionais
- **Validação:** garantir que o software atende as **expectativas** do cliente

**A verificação e validação perduram por todas as etapas do desenvolvimento** de software  
(bugs, falhas e erros podem ocorrer a qualquer momento)

# Exercício

Considerando os testes de software no desenvolvimento incremental, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Realizar testes de forma incremental fornece as seguintes vantagens: documentação do sistema facilitada, depuração simplificada, cobertura de código, teste de regressão.

II - A cobertura de código garante que cada incremento de software será testado, o que leva a uma depuração simplificada do sistema. Isto é, os testes ajudam a localizar problemas sem que seja necessário o uso de ferramentas de depuração.

III - Os testes de regressão verificam o código de forma a garantir que, conforme novos incrementos do sistema forem desenvolvidos, novos bugs não sejam criados.

IV - O teste incremental de software facilita a documentação do sistema, pois os testes servem como descrição do funcionamento do código.

- ☐ Todas as assertivas são verdadeiras.
- ☐ Somente I, III e IV.
- ☐ Somente II, III e IV.
- ☐ Somente I e II.
- ☐ Nenhuma das alternativas anteriores.

# Exercício

Considerando os testes de software no desenvolvimento incremental, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Realizar testes de forma incremental fornece as seguintes vantagens: documentação do sistema facilitada, depuração simplificada, cobertura de código, teste de regressão. **V**

[Slide 25: Desenvolvimento incremental \(dirigido a testes\)](#)

II - A cobertura de código garante que cada incremento de software será testado, o que leva a uma depuração simplificada do sistema. Isto é, os testes ajudam a localizar problemas sem que seja necessário o uso de ferramentas de depuração.

III - Os testes de regressão verificam o código de forma a garantir que, conforme novos incrementos do sistema forem desenvolvidos, novos bugs não sejam criados.

IV - O teste incremental de software facilita a documentação do sistema, pois os testes servem como descrição do funcionamento do código.

- ☐ Todas as assertivas são verdadeiras.
- ☐ Somente I, III e IV.
- ☐ Somente II, III e IV.
- ☐ Somente I e II.
- ☐ Nenhuma das alternativas anteriores.

# Exercício

Considerando os testes de software no desenvolvimento incremental, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Realizar testes de forma incremental fornece as seguintes vantagens: documentação do sistema facilitada, depuração simplificada, cobertura de código, teste de regressão. **V**

II - A cobertura de código garante que cada incremento de software será testado, o que leva a uma depuração simplificada do sistema. Isto é, os testes ajudam a localizar problemas sem que seja necessário o uso de ferramentas de depuração. **V**

[Slide 25: Desenvolvimento incremental \(dirigido a testes\)](#)

III - Os testes de regressão verificam o código de forma a garantir que, conforme novos incrementos do sistema forem desenvolvidos, novos bugs não sejam criados.

IV - O teste incremental de software facilita a documentação do sistema, pois os testes servem como descrição do funcionamento do código.

- ☐ Todas as assertivas são verdadeiras.
- ☐ Somente I, III e IV.
- ☐ Somente II, III e IV.
- ☐ Somente I e II.
- ☐ Nenhuma das alternativas anteriores.



# Exercício

Considerando os testes de software no desenvolvimento incremental, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Realizar testes de forma incremental fornece as seguintes vantagens: documentação do sistema facilitada, depuração simplificada, cobertura de código, teste de regressão. **V**

II - A cobertura de código garante que cada incremento de software será testado, o que leva a uma depuração simplificada do sistema. Isto é, os testes ajudam a localizar problemas sem que seja necessário o uso de ferramentas de depuração. **V**

III - Os testes de regressão verificam o código de forma a garantir que, conforme novos incrementos do sistema forem desenvolvidos, novos bugs não sejam criados. **V**

[Slide 25: Desenvolvimento incremental \(dirigido a testes\)](#)

IV - O teste incremental de software facilita a documentação do sistema, pois os testes servem como descrição do funcionamento do código.

- ☐ Todas as assertivas são verdadeiras.
- ☐ Somente I, III e IV.
- ☐ Somente II, III e IV.
- ☐ Somente I e II.
- ☐ Nenhuma das alternativas anteriores.



# Exercício

Considerando os testes de software no desenvolvimento incremental, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Realizar testes de forma incremental fornece as seguintes vantagens: documentação do sistema facilitada, depuração simplificada, cobertura de código, teste de regressão. **V**

II - A cobertura de código garante que cada incremento de software será testado, o que leva a uma depuração simplificada do sistema. Isto é, os testes ajudam a localizar problemas sem que seja necessário o uso de ferramentas de depuração. **V**

III - Os testes de regressão verificam o código de forma a garantir que, conforme novos incrementos do sistema forem desenvolvidos, novos bugs não sejam criados. **V**

IV - O teste incremental de software facilita a documentação do sistema, pois os testes servem como descrição do funcionamento do código. **V**

☒ Todas as assertivas são verdadeiras.

☐ Somente I, III e IV.

☐ Somente II, III e IV.

☐ Somente I e II.

☐ Nenhuma das alternativas anteriores.

[Slide 25: Desenvolvimento incremental \(dirigido a testes\)](#)

# Etapas de testes em softwares comerciais

---

- Geralmente, o sistema de software comercial tem de passar por três estágios de teste:
  - **Testes de desenvolvimento**
  - **Testes de release**
  - **Testes de usuário**

# Testes de desenvolvimento

- O sistema é testado durante o desenvolvimento
  - **Objetivo:** descobrir bugs e falhas
  - **Quem executa os testes?**
    - Projetistas do sistema
    - Programadores

É possível ter um processo de desenvolvimento em pares: um cria o sistema (programador) e o outro testa (testador)



# Testes de desenvolvimento

---

- Os testes podem ocorrer em três níveis de granularidade:
  - **Teste unitário**
  - **Teste de componentes**
  - **Teste de sistema**

# Teste unitário

- Componentes do sistema são testados isoladamente uns dos outros pelas pessoas que os desenvolveram

Cliente	Gerente	Caixa
+ idCliente + nome + CPF + Agencia + Conta + idGerente	+ id + nome + CPF + Agencia	+ id + nome + CPF + Agencia
+ criarConta() + alterarSenha() + consultarExtrato() + sacarDinheiro()	+ criarConta() + alterarSenha()	+ sacarDinheiro()

# Teste unitário

- Componentes do sistema são testados isoladamente uns dos outros pelas pessoas que os desenvolveram

Componentes podem ser entidades simples (classes, objetos) ou agrupamentos dessas entidades

Cliente	Gerente	Caixa
+ idCliente + nome + CPF + Agencia + Conta + idGerente	+ id + nome + CPF + Agencia	+ id + nome + CPF + Agencia
+ criarConta() + alterarSenha() + consultarExtrato() + sacarDinheiro()	+ criarConta() + alterarSenha()	+ sacarDinheiro()

# Teste unitário

- Componentes do sistema são testados isoladamente uns dos outros pelas pessoas que os desenvolveram

Componentes podem ser entidades simples (classes, objetos) ou agrupamentos dessas entidades

O desenvolvimento dos componentes e os seus testes são feitos de forma intercalada

Cliente	Gerente	Caixa
+ idCliente + nome + CPF + Agencia + Conta + idGerente	+ id + nome + CPF + Agencia	+ id + nome + CPF + Agencia
+ criarConta() + alterarSenha() + consultarExtrato() + sacarDinheiro()	+ criarConta() + alterarSenha()	+ sacarDinheiro()

# Teste unitário

- Componentes do sistema são testados isoladamente uns dos outros pelas pessoas que os desenvolveram

**Testes unitários devem,  
sempre que possível,  
ser automatizados**


Cliente	Gerente	Caixa
+ idCliente + nome + CPF + Agencia + Conta + idGerente	+ id + nome + CPF + Agencia	+ id + nome + CPF + Agencia
+ criarConta() + alterarSenha() + consultarExtrato() + sacarDinheiro()	+ criarConta() + alterarSenha()	+ sacarDinheiro()



# Teste unitário

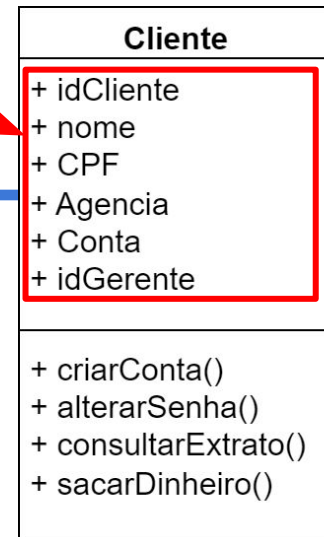
- Quando um teste unitário está sendo elaborado, precisamos:
  - Testar todas as operações associadas ao objeto (**métodos**)

Cliente
+ idCliente + nome + CPF + Agencia + Conta + idGerente
+ criarConta() + alterarSenha() + consultarExtrato() + sacarDinheiro()



# Teste unitário

- Quando um teste unitário está sendo elaborado, precisamos:
  - Testar todas as operações associadas ao objeto (**métodos**)
  - Definir e verificar o valor de todos os **atributos**
    - Incluindo os valores limítrofes (máximo e mínimo)



# Teste unitário

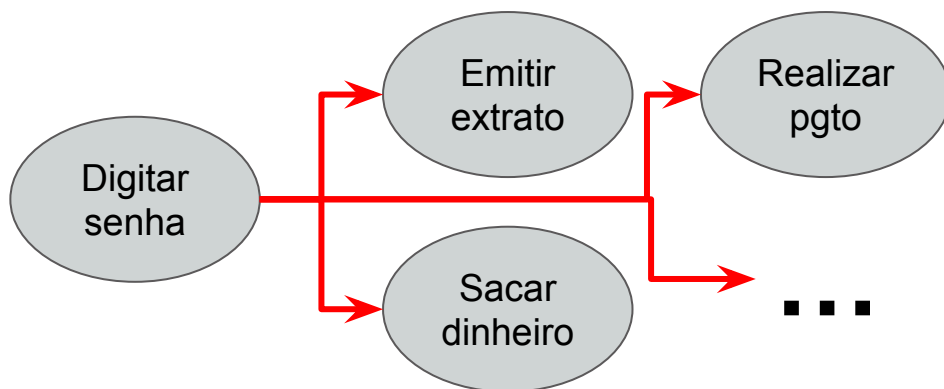
- Quando um teste unitário está sendo elaborado, precisamos:
  - Testar o objeto em **todos os estados possíveis**, o que significa simular todos os eventos que causam mudanças de estado



Cliente
+ idCliente + nome + CPF + Agencia + Conta + idGerente
+ criarConta() + alterarSenha() + consultarExtrato() + sacarDinheiro()

# Teste unitário

- Quando um teste unitário está sendo elaborado, precisamos:
  - Testar o objeto em **todos os estados possíveis**, o que significa simular todos os eventos que causam mudanças de estado



Cliente
+ idCliente + nome + CPF + Agencia + Conta + idGerente
+ criarConta() + alterarSenha() + consultarExtrato() + sacarDinheiro()

# Testes unitários automatizados

---

- Existem frameworks para automatização de testes (**ex:** JUnit)
  - Fornecem classes de testes genéricas que permitem que criemos casos de testes específicos
  - Executam todos os casos de testes automaticamente, e mostram o resultado de cada teste na tela

# Testes unitários automatizados

---

- Existem frameworks para automatização de testes (**ex:** JUnit)
  - Fornecem classes de testes genéricas que permitem que criemos casos de testes específicos
  - Executam todos os casos de testes automaticamente, e mostram o resultado de cada teste na tela
- **Vantagem:** permite testar o componente rapidamente, sempre que alguma mudança ou correção for feita nele

# Testes unitários automatizados - Exemplo Python

Classe de testes  
contém os **casos  
de testes**  
(funções “test\_”)

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

# Testes unitários automatizados - Exemplo Python

Cada função é um  
caso de teste  
(Etapa de  
Configuração)

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```



# Testes unitários automatizados - Exemplo Python

Cada função é executada uma após a outra

Se qualquer função falhar, então o teste falhou

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

# Testes unitários automatizados - Exemplo Python

Uma função falha se  
uma das afirmações  
(ou **asserts**) falhar

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

# Testes unitários automatizados - Exemplo Python

**Ex:**  
`assertEqual(p1, p2)`  
falha se o `p1 != p2`

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

# Testes unitários automatizados - Exemplo Python

**Ex:**  
`assertEqual(p1, p2)`  
falha se o `p1 != p2`

P1 é chamado de  
**Entrada**  
(chamada)

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

# Testes unitários automatizados - Exemplo Python

**Ex:**  
`assertEqual(p1, p2)`  
falha se o `p1 != p2`

P1 é chamado de  
**Entrada**  
(chamada)

P2 é chamado de  
**Saída**  
(resultado esperado)

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

# Testes unitários automatizados - Exemplo Python

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

**Ex:**  
`assertTrue(p1)` falha se  
o `p1 != True`

**Ex:**  
`assertFalse(p1)` falha  
se o `p1 != False`

# Nomenclatura de Testes unitários automatizados

---

- **Configuração:** sistema é iniciado com um caso de teste (entradas + saída esperada)
- **Chamada:** o componente (função, classe, etc) a ser testado é executado (chamado)
- **Afirmação (*assert*):** comparação do resultado da chamada com o resultado esperado no caso de testes. Se a afirmação for verdadeira, o teste foi bem sucedido. Se falsa, o teste falhou!

## *Mock object* em Testes unitários automatizados

---

- Às vezes um componente precisa de outro para funcionar (**ex:** Cliente precisa acessar Banco de Dados do Banco para consultar extrato)
- As vezes, o componente que precisamos não está completamente desenvolvido/finalizado



# *Mock object* em Testes unitários automatizados

---

- Às vezes um componente precisa de outro para funcionar (**ex:** Cliente precisa acessar Banco de Dados do Banco para consultar extrato)
- As vezes, o componente que precisamos não está completamente desenvolvido/finalizado
- As vezes, este componente possui acesso lento (como o banco de dados), ou podemos precisar testar eventos raros deste componente (ex: falha no acesso ao banco de dados)

Nesses casos, criamos um ***mock object***

# Definição de *Mock object*

---

- São objetos com a mesma interface que os objetos externos usados para simular sua funcionalidade
  - **Ex:** mock object de banco de dados simula o acesso a um banco de dados

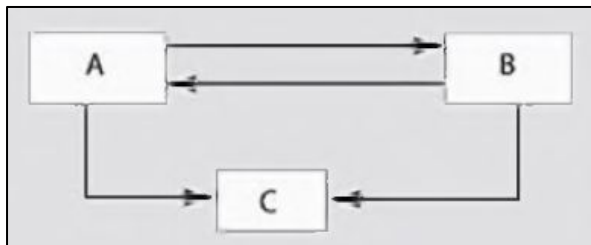
# Definição de *Mock object*

---

- São objetos com a mesma interface que os objetos externos usados para simular sua funcionalidade
  - Podem ser construídas usando estruturas de dados para tornar os testes mais rápidos (**Ex:** mock object de banco de dados pode ser um vetor de dados)
  - Permitem testar eventos raros (falha no DB, por exemplo)

# Testes de componentes

- Componentes de software são compostos por objetos que interagem
  - Erros de interface no componente aparecem em decorrência de interações entre os objetos do componente
- Testes de componentes **tentam demonstrar que a interface do componente** se comporta de acordo com sua especificação



# Exercício

---

Considerando o escopo de testes de software, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Testes unitários devem verificar cada atributo ou método, e também avaliar todos os estados possíveis de um objeto.

II - O teste componentes permite testar componentes isolados uns dos outros, de forma automática.

III - Os testes de desenvolvimento em sistemas comerciais geralmente tem três estágios: testes de desenvolvimento, testes de release e testes de usuário.

IV - Dentre os testes de desenvolvimento, há o teste subintegração, teste de componentes e de sistema.

- ☐ Somente I e II.
- ☐ Somente II e III.
- ☐ Somente IV.
- ☐ Somente I e III.
- ☐ Somente I, II e III.

# Exercício

Considerando o escopo de testes de software, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Testes unitários devem verificar cada atributo ou método, e também avaliar todos os estados possíveis de um objeto. **V** [Slide 47: Teste unitário](#) e [Slide 49](#)

II - O teste componentes permite testar componentes isolados uns dos outros, de forma automática.

III - Os testes de desenvolvimento em sistemas comerciais geralmente tem três estágios: testes de desenvolvimento, testes de release e testes de usuário.

IV - Dentre os testes de desenvolvimento, há o teste subintegração, teste de componentes e de sistema.

- ☐ Somente I e II.
- ☐ Somente II e III.
- ☐ Somente IV.
- ☐ Somente I e III.
- ☐ Somente I, II e III.

# Exercício

Considerando o escopo de testes de software, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Testes unitários devem verificar cada atributo ou método, e também avaliar todos os estados possíveis de um objeto. **V**

II - O teste componentes permite testar componentes isolados uns dos outros, de forma automática. **F** [Slide 44: Teste unitário](#) e [Slide 65](#)

III - Os testes de desenvolvimento em sistemas comerciais geralmente tem três estágios: testes de desenvolvimento, testes de release e testes de usuário.

IV - Dentre os testes de desenvolvimento, há o teste subintegração, teste de componentes e de sistema.

- ☐ Somente I e II.
- ☐ Somente II e III.
- ☐ Somente IV.
- ☐ Somente I e III.
- ☐ Somente I, II e III.

# Exercício

Considerando o escopo de testes de software, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Testes unitários devem verificar cada atributo ou método, e também avaliar todos os estados possíveis de um objeto. **V**

II - O teste componentes permite testar componentes isolados uns dos outros, de forma automática. **F**

III - Os testes de desenvolvimento em sistemas comerciais geralmente tem três estágios: testes de desenvolvimento, testes de release e testes de usuário. **V**

IV - Dentre os testes de desenvolvimento, há o teste subintegração, teste de componentes e de sistema.

- ☐ Somente I e II.
- ☐ Somente II e III.
- ☐ Somente IV.
- ☐ Somente I e III.
- ☐ Somente I, II e III.

[Slide 39: Etapas de testes em softwares comerciais](#)



# Exercício

Considerando o escopo de testes de software, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Testes unitários devem verificar cada atributo ou método, e também avaliar todos os estados possíveis de um objeto. **V**

II - O teste componentes permite testar componentes isolados uns dos outros, de forma automática. **F**

III - Os testes de desenvolvimento em sistemas comerciais geralmente tem três estágios: testes de desenvolvimento, testes de release e testes de usuário. **V**

IV - Dentre os testes de desenvolvimento, há o teste subintegração teste de componentes e de sistema. **F**

[Slide 41: Testes de desenvolvimento](#)

☐ Somente I e II.

☐ Somente II e III.

☐ Somente IV.

☒ Somente I e III.

☐ Somente I, II e III.

# Classificação de Erros de Interface

---

- Erros de interface são classificados nas seguintes classes:
  - **Mau uso de interface**
  - **Mau-entendimento de interface**
  - **Erros de timing**

# Mau uso de Interface

- Componente chamador **conhece o comportamento esperado** do outro componente porém **comete um erro no uso** de sua interface
- **Ex:** função chamada com parâmetros com
  - Tipo errado
  - Ordem errada
  - Número errado de parâmetros

```
#include <stdio.h>

int sum(int a, int b)
{
    return a + b;
}

int main()
{
    int add = sum("a", "b");

    printf("Sum is: %d", add);

    return 0;
}
```

Formal Parameter

Actual Parameter

# Mau-entendimento de interface

- Componente chamador **desconhece a especificação** da interface do componente chamado e **faz suposições** sobre seu comportamento
- **Ex:** função de busca binária chamada com um vetor não ordenado

0	1	2	3	4	5	6	7	8	9
2	5	8	12	25	23	38	56	72	91
0	1	2	3	4	5	6	7	8	9
2	5	8	12	25	23	38	56	72	91

**Ex:** Buscar por 23

Divida o vetor na metade.

Como  $23 < 25$ , procure por 23 na primeira metade do vetor!

# Mau-entendimento de interface

- Componente chamador **desconhece a especificação** da interface do componente chamado e **faz suposições** sobre seu comportamento
- **Ex:** função de busca binária chamada com um vetor não ordenado

0	1	2	3	4	5	6	7	8	9
2	5	8	12	25	23	38	56	72	91
0	1	2	3	4	5	6	7	8	9
2	5	8	12	25	23	38	56	72	91

Neste caso somos induzidos a procurar pelo **23** na metade errada do vetor, pois supomos que o vetor estava ordenado (e não é o caso!)

# Erros de *timing*

---

- Ocorrem em sistemas em tempo real que usam uma memória compartilhada ou uma interface de passagem de mensagens
- O produtor e o consumidor de dados podem operar em velocidades diferentes
  - **Resultado:** o consumidor acessa informações desatualizadas

## Exemplo de Erros de *timing*

---

- **Ex:** programa para envio dos emails de um provedor (ex: GMAIL)
  - Verifica a caixa de saída a cada 10 min
  - Porém ele não verifica se a caixa de saída está vazia
  - **Resultado:** SEGFAULT



# Diretrizes para testes de componentes (interface)

---

- Examinar código explicitamente, projetando casos de teste com valores de parâmetros extremos
  - **Ex:** função para calcular números primos ( $p < 0$ ,  $p = 0$ ,  $p > 0$ )
- Sempre testar componentes cujos parâmetros são ponteiros com o valor NULO
  - **Ex:** passar um vetor nulo para uma função *soma(\*vetor)*



# Diretrizes para testes de componentes (interface)

---

- Projete testes de estresse para descobrir erros de *timing* em sistemas complexos
  - **Ex:** em uma caixa de email, projete um cenário onde muitos emails são enviados ao mesmo tempo para a caixa de saída (e veja se algum erro ocorre)
    - A caixa fica cheia? Há estouro da fila? O programa sofre *crash* com *segmentation fault*?)



# Diretrizes para testes de componentes (interface)

---

- No caso de componentes que compartilham memória, teste se a ordem da execução deles causa algum problema
  - **Ex:** se eu executar primeiro o programa para enviar emails da caixa de saída, será que isso gera algum problema?



# Diretrizes para testes de componentes (interface)

---

- No caso de componentes que compartilham memória, teste se a ordem da execução deles causa algum problema
  - **Ex:** se eu executar primeiro o programa para enviar emails da caixa de saída, será que isso gera algum problema?
  - Em teoria o programa deve verificar se há emails para serem enviados, antes de qualquer ação
  - E se eu executar primeiro o programa para colocar um email na caixa de saída?



Gmail

# Testes de sistema

---

- Testes realizados com os componentes do sistema integrados, compondo o sistema completo
  - **Objetivos:**
    - Encontrar erros nas interações entre componentes e problemas de interface entre componentes
    - Atestar que o sistema cumpre com seus requisitos (funcionais e não-funcionais)

# Testes de sistema

---

- Sistemas grandes podem exigir **processos multi-estágio**
  - **Processos multi-estágio:** componentes são integrados para formar subsistemas
    - Subsistemas são testados individualmente antes de serem combinados para compor o sistema final

# Testes de sistema x Testes de Componentes

---

- **Teste de sistema verificam todos os componentes** (incluindo os reusáveis e de prateleira)
  - Testes de componente testam componentes recém desenvolvidos
- Testes de sistema **verificam a integração** entre componentes desenvolvidos por diferentes membros da equipe ou grupos
  - Testes de componentes nem sempre verificam a integração

# Testes de sistema

---

- Verificam o comportamento do sistema como um todo (**comportamento emergente**)
- **Ex:** Integração de componentes de autenticação e atualização de dados.
  - Permite testar um sistema no qual há restrição na atualização de informações, permitida apenas para usuários autorizados

# Testes de sistema

---

- Verificam o comportamento do sistema como um todo (**comportamento emergente**)

**Lembrete:**

Devemos testar apenas os recursos que estão previstos na especificação de requisitos do sistema

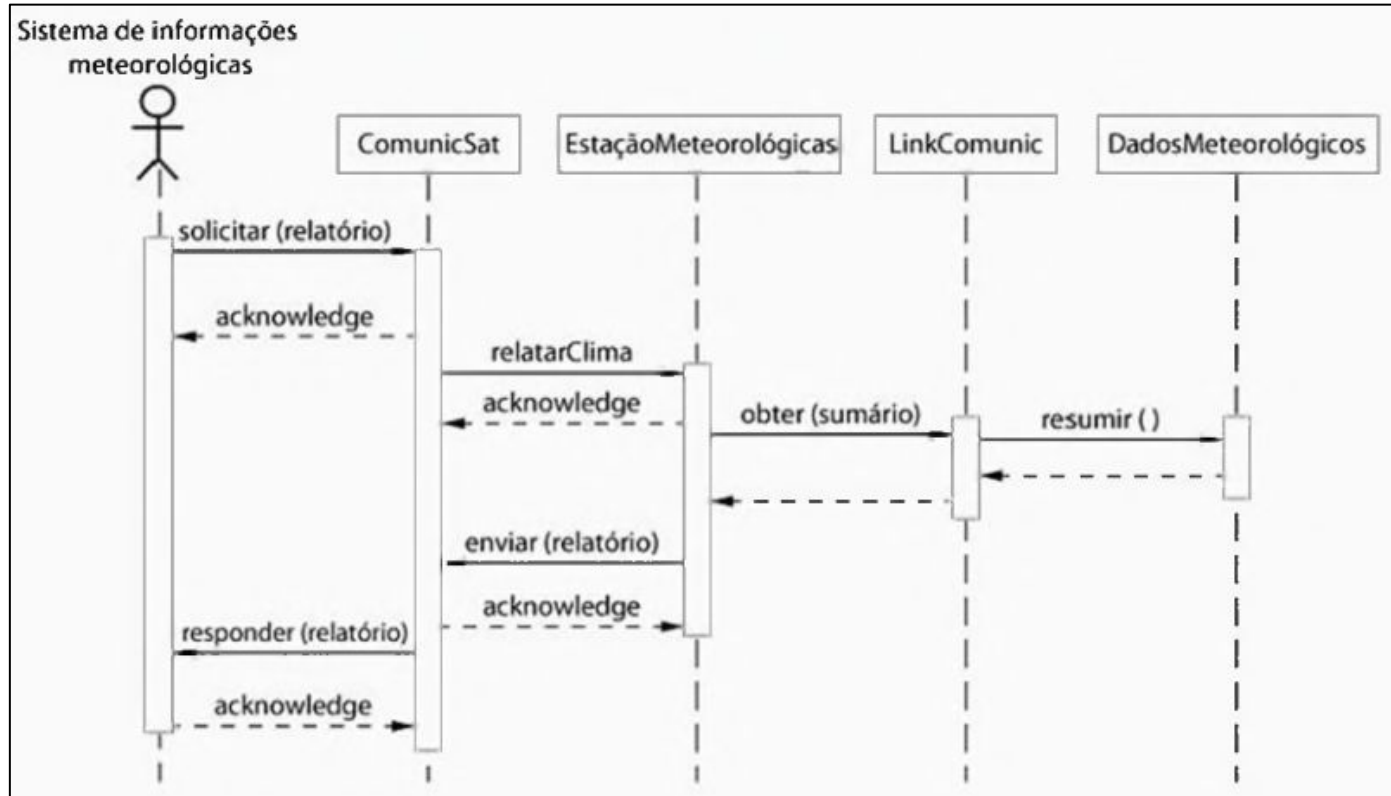


# Testes de sistema

---

- Também permitem identificar equívocos dos desenvolvedores de componentes sobre outros componentes do sistema.
  - Pressuposições sobre a interface dos componentes
  - Mau uso da interface
  - E demais falhas que também são detectadas pelos testes de componentes
- Testes de sistema são úteis também para testes de desempenho e confiabilidade

## Caso de uso e diagramas de sequência são abordagens eficazes para organizar os **testes de sistema**



# Testes de sistema

---

- Testes baseados em caso de uso e diagramas de sequência são abordagens eficazes para testes de sistema
  - Descrevem a interação entre componentes
  - Facilitam a visualização do comportamento esperado do sistema (requisitos, entradas e saídas)

# Exercício

Considerando o escopo de testes de componentes, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Os testes de componentes verificam erros de interface, que podem surgir em decorrência de interações entre componentes.

II - Um dos tipos de erros de interface é o mau uso da interface. Este erro ocorre pois componente chamador de outro componente desconhece a especificação da interface deste (ex: chamar uma função que necessita de um vetor ordenado passando como parâmetro um vetor desordenado).

III - Outro tipo de erro de interface é o mau-entendimento da interface. Este erro ocorre pois componente chamador de outro componente conhece o comportamento esperado do componente, porém comete um erro ao utilizar sua interface (ex: chamada de função com parâmetros errados).

IV - O último tipo de erro de interface são os erros de timing. Eles ocorrem em sistemas em tempo real que usam uma memória compartilhada ou uma interface para passagem de mensagens.

- ☐ Todas as assertivas são verdadeiras.
- ☐ Somente II e III.
- ☐ Somente I e IV.
- ☐ Somente I, II e III.
- ☐ Nenhuma das alternativas anteriores.

# Exercício

Considerando o escopo de testes de componentes, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Os testes de componentes verificam erros de interface, que podem surgir em decorrência de interações entre componentes. **V**

II - Um dos tipos de erros de interface é o mau uso da interface. Este erro ocorre pois componente chamador de outro componente desconhece a especificação da interface deste (ex: chamar uma função que necessita de um vetor ordenado passando como parâmetro um vetor desordenado).

III - Outro tipo de erro de interface é o mau-entendimento da interface. Este erro ocorre pois componente chamador de outro componente conhece o comportamento esperado do componente, porém comete um erro ao utilizar sua interface (ex: chamada de função com parâmetros errados).

IV - O último tipo de erro de interface são os erros de timing. Eles ocorrem em sistemas em tempo real que usam uma memória compartilhada ou uma interface para passagem de mensagens.

- ☐ Todas as assertivas são verdadeiras.
- ☐ Somente II e III.
- ☐ Somente I e IV.
- ☐ Somente I, II e III.
- ☐ Nenhuma das alternativas anteriores.

# Exercício

Considerando o escopo de testes de componentes, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Os testes de componentes verificam erros de interface, que podem surgir em decorrência de interações entre componentes. **V**

II - Um dos tipos de erros de interface é o mau uso da interface. Este erro ocorre pois componente chamador de outro componente desconhece a especificação da interface deste (ex: chamar uma função que necessita de um vetor ordenado passando como parâmetro um vetor desordenado). **F**

III - Outro tipo de erro de interface é o mau-entendimento da interface. Este erro ocorre pois componente chamador de outro componente conhece o comportamento esperado do componente, porém comete um erro ao utilizar sua interface (ex: chamada de função com parâmetros errados).

IV - O último tipo de erro de interface são os erros de timing. Eles ocorrem em sistemas em tempo real que usam uma memória compartilhada ou uma interface para passagem de mensagens.

- ☐ Todas as assertivas são verdadeiras.
- ☐ Somente II e III.
- ☐ Somente I e IV.
- ☐ Somente I, II e III.
- ☐ Nenhuma das alternativas anteriores.

# Exercício

Considerando o escopo de testes de componentes, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Os testes de componentes verificam erros de interface, que podem surgir em decorrência de interações entre componentes. **V**

II - Um dos tipos de erros de interface é o mau uso da interface. Este erro ocorre pois componente chamador de outro componente desconhece a especificação da interface deste (ex: chamar uma função que necessita de um vetor ordenado passando como parâmetro um vetor desordenado). **F**

III - Outro tipo de erro de interface é o mau-entendimento da interface. Este erro ocorre pois componente chamador de outro componente conhece o comportamento esperado do componente, porém comete um erro ao utilizar sua interface (ex: chamada de função com parâmetros errados). **F**

IV - O último tipo de erro de interface são os erros de timing. Eles ocorrem em sistemas em tempo real que usam uma memória compartilhada ou uma interface para passagem de mensagens.

- ☐ Todas as assertivas são verdadeiras.
- ☐ Somente II e III.
- ☐ Somente I e IV.
- ☐ Somente I, II e III.
- ☐ Nenhuma das alternativas anteriores.

# Exercício

Considerando o escopo de testes de componentes, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Os testes de componentes verificam erros de interface, que podem surgir em decorrência de interações entre componentes. **V**

II - Um dos tipos de erros de interface é o mau uso da interface. Este erro ocorre pois componente chamador de outro componente desconhece a especificação da interface deste (ex: chamar uma função que necessita de um vetor ordenado passando como parâmetro um vetor desordenado). **F**

III - Outro tipo de erro de interface é o mau-entendimento da interface. Este erro ocorre pois componente chamador de outro componente conhece o comportamento esperado do componente, porém comete um erro ao utilizar sua interface (ex: chamada de função com parâmetros errados). **F**

IV - O último tipo de erro de interface são os erros de timing. Eles ocorrem em sistemas em tempo real que usam uma memória compartilhada ou uma interface para passagem de mensagens. **V**

- ☐ Todas as assertivas são verdadeiras.
- ☐ Somente II e III.
- ☒ Somente I e IV.
- ☐ Somente I, II e III.
- ☐ Nenhuma das alternativas anteriores.



# Testes de release

---

- Uma versão completa do sistema é testada
  - **Objetivo:** verificar se o sistema atende aos requisitos dos ***stakeholders do sistema*** (usuários e demais interessados na operação do sistema)
  - **Quem executa os testes?**
    - Equipe de teste independente (que não esteve envolvida com o desenvolvimento do sistema)

# Teste de release vs Teste de sistema

---

- Testes de release são **testes de validação**
  - **Objetivo:** convencer o fornecedor do sistema de que o sistema é bom o suficiente para uso

# Teste de release vs Teste de sistema

---

- Testes de release são **testes de validação**
  - **Objetivo:** convencer o fornecedor do sistema de que o sistema é bom o suficiente para uso
    - Atende aos seus requisitos (de sistema e usuários finais)
    - Mostrar que o sistema não falha durante o uso normal, fornecendo funcionalidade, desempenho, e confiança

# Teste de release vs Teste de sistema

---

- Testes de release são **testes de validação**
  - **Objetivo:** convencer o fornecedor do sistema de que o sistema é bom o suficiente para uso
    - Atende aos seus requisitos (de sistema e usuários finais)
    - Mostrar que o sistema não falha durante o uso normal, fornecendo funcionalidade, desempenho, e confiança
- Testes de sistema são **testes de defeitos**
  - **Objetivo:** encontrar bugs / falhas

# Testes de release

---

- São feitos considerando que o sistema é um caixa-preta
  - Comportamento do sistema só pode ser previsto quando fornecemos uma entrada para ele
  - Para cada entrada, há uma saída
  - O foco está na funcionalidade do sistema, e não na sua implementação
  - Esta forma de testar é chamada de **teste funcional**

# Classificação de Testes de release

---

- **Testes baseados em requisitos**
  - Cada requisito é construído pensando em um conjunto de testes a ser executado
- **Testes de cenário**
  - Imaginar cenários típicos de uso e os usa para desenvolver casos de teste para o sistema
- **Testes de desempenho**
  - Ter certeza que o sistema consegue processar a carga a que se destina

# Testes de release baseados em requisitos

---

- Abordagem sistemática para projeto de casos de teste
- Cada requisito deriva de um conjunto de testes, projetado especificamente para validar o requisito
- São **testes de validação**
- **Ex:** sistema de prescrições médicas com os seguintes requisitos:

Se um paciente é alérgico a algum medicamento específico, uma prescrição para esse medicamento deve resultar em uma mensagem de aviso.

Se um médico opta por ignorar um aviso, ele deve justificar sua decisão.

# Testes de cenário

---

- Consiste em imaginar **cenários típicos de uso** e usa-los para desenvolver casos de teste para o sistema
  - **Cenário de uso:** uma história que descreve uma maneira de usar o sistema



# Testes de cenário

---

- Consiste em imaginar **cenários típicos de uso** e usa-los para desenvolver casos de teste para o sistema
  - **Cenário de uso:** uma história que descreve uma maneira de usar o sistema
    - Cenários devem ser realistas, e usuários reais do sistema devem ser capazes de se relacionar com eles
  - Permite testar vários requisitos dentro de um mesmo cenário de uso

# Testes de cenário

---

- **Ex:** A enfermeira Kate vai utilizar o sistema de prescrição médica que vimos nos exemplos anteriores
  - Conforme Kate utiliza o sistema, ela irá cometer erros
  - Devemos anotar os erros, e o comportamento do sistema em resposta a esses erros
    - “O sistema apresentou falha?”
    - “O sistema teve desempenho aceitável?”
    - “Kate acessou áreas do sistema que não deveria?”

# Testes de desempenho

---

- Assegurar que o sistema consegue processar a carga a que se destina
- Consiste em executar uma série de testes em que você aumenta a carga até que o desempenho do sistema se torne inaceitável
  - **Objetivo:**
    - Demonstrar que o sistema atende seus requisitos
    - Descobrir problemas e defeitos do sistema

# Testes de desempenho

---

- Para testar se os requisitos de desempenho estão sendo alcançados, você pode ter de construir um **perfil operacional**.
  - **Perfil operacional**: análise do funcionamento do sistema que reflete o ambiente de trabalho real ao qual ele será submetido

# Exemplo de análise de perfil operacional

---

- Seja um sistema que possui as seguintes características de funcionamento:
  - 90% dos arquivos salvos no HDD possuem no máximo 4 MB (**grupo A**)
  - 7% possuem até 2 MB (**grupo B**)
  - 3% possuem menos que 512 KB (**grupo C**)
- Nesse caso você tem de projetar o perfil operacional para que a maioria dos testes seja do **grupo A**

# Testes de desempenho

---

- Construir perfis operacionais não é necessariamente a melhor escolha para testes de desempenho
  - **Estressar o sistema** se mostrou mais eficiente

# Testes de desempenho

---

- Construir perfis operacionais não é necessariamente a melhor escolha para testes de desempenho
  - **Estressar o sistema** se mostrou mais eficiente
    - Projetar testes para os limites do sistema, fazendo demandas que estejam fora dos limites de projeto do software
    - **Ex:** seja um banco de dados, projetado para atender até 200 usuários ao mesmo tempo
    - Se conseguirmos mostrar que ele atende 300 usuários, esse banco de dados passou no teste de desempenho

# Testes de usuário

---

- Os usuários ou potenciais usuários de um sistema testam o sistema
  - Pode conter testes formais (definido previamente) ou informais (cada usuário é livre para explorar o sistema)
  - **Objetivo:** decidir se o sistema deve ser aceito ou se é necessário um desenvolvimento adicional
  - **Quem executa os testes?**
    - Usuários



# Testes de usuário

---

- **Porque testes de usuário são importantes?**
  - O desenvolvedor, por mais que se esforce, não consegue replicar o cenário de uso real do sistema
    - Testes no ambiente do desenvolvedor são inevitavelmente artificiais
    - **Ex:** Sistema hospitalar é usado em um ambiente clínico, com atividades imprevisíveis ocorrendo ao mesmo tempo (emergências, exames, procedimentos)

# Classificação de Testes de usuário

---

- **Testes alfa**

- Usuários e equipe de desenvolvimento testam o software no local do desenvolvedor

- **Testes beta**

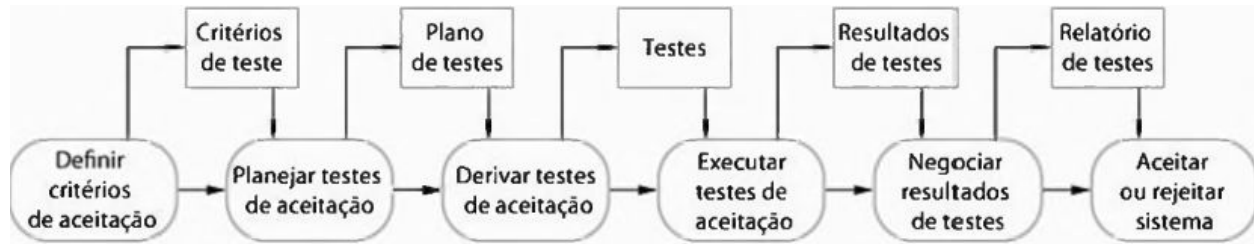
- Release do software é disponibilizado aos usuários para que possam experimentar e relatar problemas aos desenvolvedores

- **Testes de aceitação**

- Clientes testam um sistema para decidir se está ou não pronto para ser aceito e implantado no ambiente do cliente (produção)

# Etapas dos Testes de aceitação

- **Definir critérios de aceitação:** requisitos para o aceite do software
- **Planejar testes de aceitação:** recursos e orçamento para os testes
- **Derivar testes de aceitação:** projeto dos testes de aceitação
- **Executar testes de aceitação:** executar testes de aceitação
- **Negociar resultados de teste:** que erros o cliente tolera no sistema?
- **Aceitar/rejeitar sistema:** decisão dos clientes e desenvolvedores



# Depuração de software

---

- Caso um sistema apresente falhas em um dos testes, ele precisará passar por processos de **depuração**
- A **depuração** consiste em técnicas e processos para correção de falhas de software, tais como:
  - **Depuração por Força bruta (*brute force*)**
  - **Rastreamento (*backtracking*)**
  - **Eliminação da causa**

# Depuração por Força bruta (*Brute force*)

---

- “Deixe o computador encontrar a falha (erro)”
  - Análise de logs e despejos de memória e demais informações na esperança de encontrar a falha
- **Problema:**
  - Processo muito lento
  - Grande volume de informações
  - Falha pode não ser encontrada



# Depuração por Rastreamento (*Backtracking*)

---

- O código-fonte é investigado a partir do ponto onde a falha ou sintoma foi detectada até que a causa seja encontrada
  - Busca retroativa pelo código
- **Vantagem:**
  - Maior chance de encontrar a falha
- **Problema:**
  - Processo lento em softwares grandes

# Depuração por Eliminação da causa

---

- É preparada uma lista de todas as causas possíveis (**hipóteses de causa**)
- São realizados testes para eliminar cada uma das hipóteses
  - Para isso, é preciso deduzir as possíveis origens da falha
- **Vantagem:**
  - Maior chance de encontrar a falha
- **Problema:**
  - Requer vários testes de hipóteses de causa

# Exercício

Considerando os testes de release, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Na abordagem de testes de release baseados em requisitos, cada conjunto de testes é construído para validar um requisito.

II - Na abordagem de testes de cenário, cada caso de teste é utilizado para construir cenários típicos de uso do sistema.

III - Na abordagem de testes de desempenho, o objetivo é garantir o sistema possui alto desempenho.

IV - Os testes de release baseados em requisitos, são testes de defeitos.

- ☐ Somente I.
- ☐ Somente I e II.
- ☐ Somente I, II e IV.
- ☐ Somente III.
- ☐ Nenhuma das assertivas é verdadeira.



# Exercício

Considerando os testes de release, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Na abordagem de testes de release baseados em requisitos, cada conjunto de testes é construído para validar um requisito. **F**

II - Na abordagem de testes de cenário, cada caso de teste é utilizado para construir cenários típicos de uso do sistema.

III - Na abordagem de testes de desempenho, o objetivo é garantir o sistema possui alto desempenho.

IV - Os testes de release baseados em requisitos, são testes de defeitos.

- ☐ Somente I.
- ☐ Somente I e II.
- ☐ Somente I, II e IV.
- ☐ Somente III.
- ☐ Nenhuma das assertivas é verdadeira.

# Exercício

Considerando os testes de release, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Na abordagem de testes de release baseados em requisitos, cada conjunto de testes é construído para validar um requisito. **F**

II - Na abordagem de testes de cenário, cada caso de teste é utilizado para construir cenários típicos de uso do sistema. **F**

III - Na abordagem de testes de desempenho, o objetivo é garantir o sistema possui alto desempenho.

IV - Os testes de release baseados em requisitos, são testes de defeitos.

- ☐ Somente I.
- ☐ Somente I e II.
- ☐ Somente I, II e IV.
- ☐ Somente III.
- ☐ Nenhuma das assertivas é verdadeira.

# Exercício

Considerando os testes de release, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Na abordagem de testes de release baseados em requisitos, cada conjunto de testes é construído para validar um requisito. **F**

II - Na abordagem de testes de cenário, cada caso de teste é utilizado para construir cenários típicos de uso do sistema. **F**

III - Na abordagem de testes de desempenho, o objetivo é garantir o sistema possui alto desempenho. **F**

IV - Os testes de release baseados em requisitos, são testes de defeitos.

- ☐ Somente I.
- ☐ Somente I e II.
- ☐ Somente I, II e IV.
- ☐ Somente III.
- ☐ Nenhuma das assertivas é verdadeira.

# Exercício

Considerando os testes de release, marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Na abordagem de testes de release baseados em requisitos, cada conjunto de testes é construído para validar um requisito. **F**

II - Na abordagem de testes de cenário, cada caso de teste é utilizado para construir cenários típicos de uso do sistema. **F**

III - Na abordagem de testes de desempenho, o objetivo é garantir o sistema possui alto desempenho. **F**

IV - Os testes de release baseados em requisitos, são testes de defeitos. **F**

- ☐ Somente I.
- ☐ Somente I e II.
- ☐ Somente I, II e IV.
- ☐ Somente III.
- ☒ Nenhuma das assertivas é verdadeira.



# Referencial Bibliográfico

---

- SOMMERVILLE, Ian. **Engenharia de Software**. 6. ed. São Paulo: Addison-Wesley, 2003.
- PRESSMAN, Roger S. **Engenharia de Software**. São Paulo: Makron Books, 1995.
- JUNIOR, H. E. **Engenharia de Software na Prática**. Novatec, 2010.