Web Academy

Testes

Ufac

Manoel Limeira de Lima Júnior

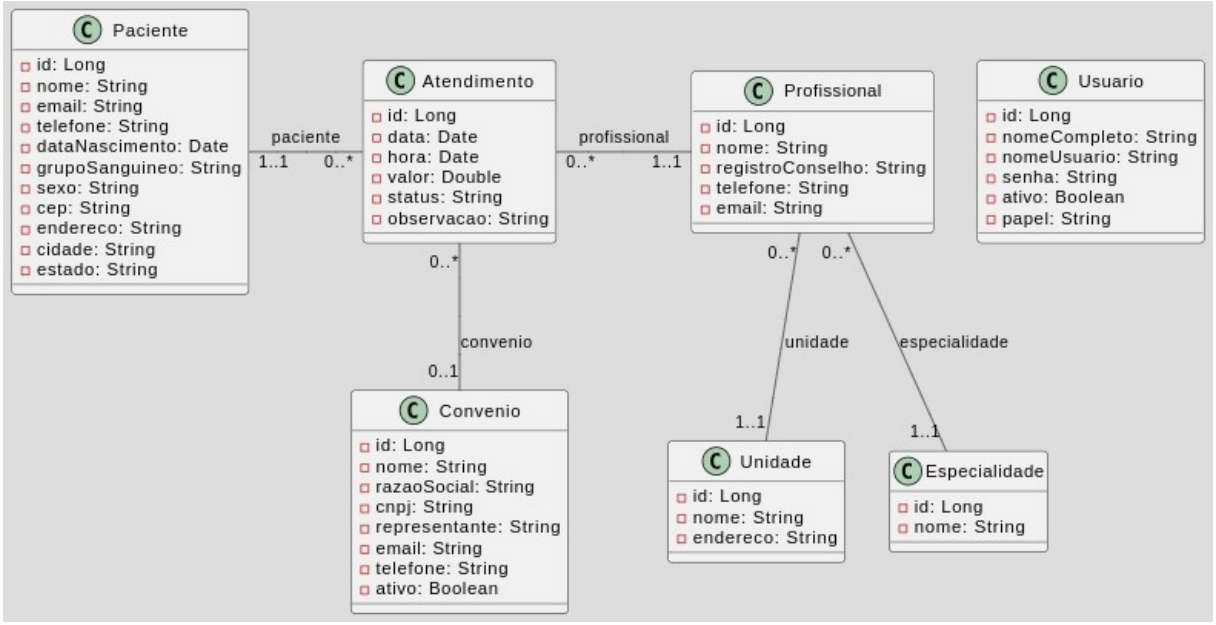
Web Academy

Ufac

Apresentação

SGCM – Sistema de Gerenciamento de Consultas Médicas

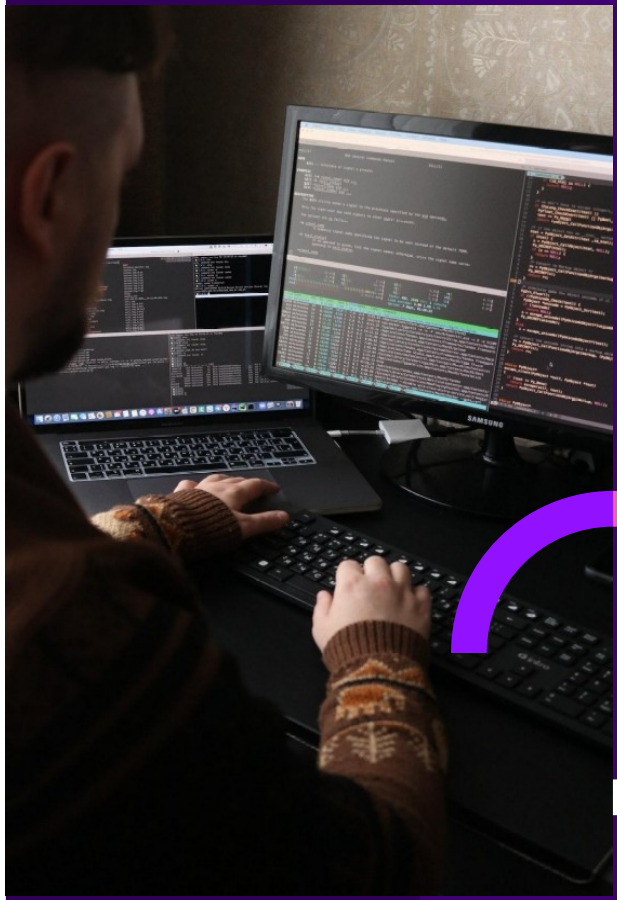
- Documentação: https://github.com/webacademyufac/sgcmdocs
  - Diagrama de classes

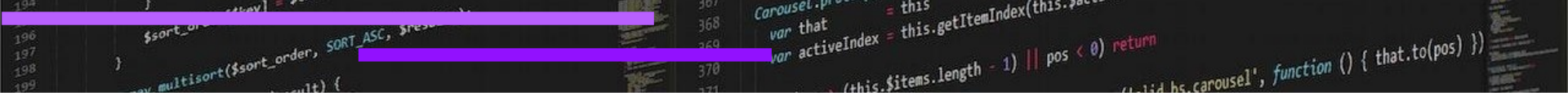


Web Academy

Ementa

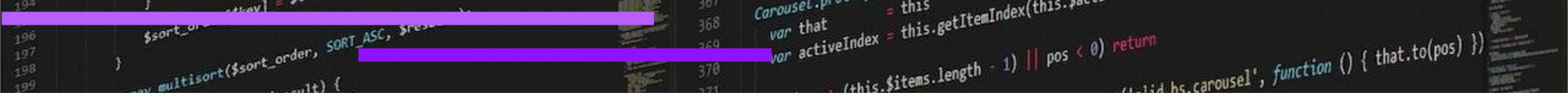
1. Visão geral.
2. Taxonomia (testes de unidade, integração e sistema).
3. Testes Automatizados.
4. Teste de back-end (teste de API REST) e de front-end (testes de UI, E2E).
5. Frameworks de teste (back-end e front-end).
6. Cobertura de código.





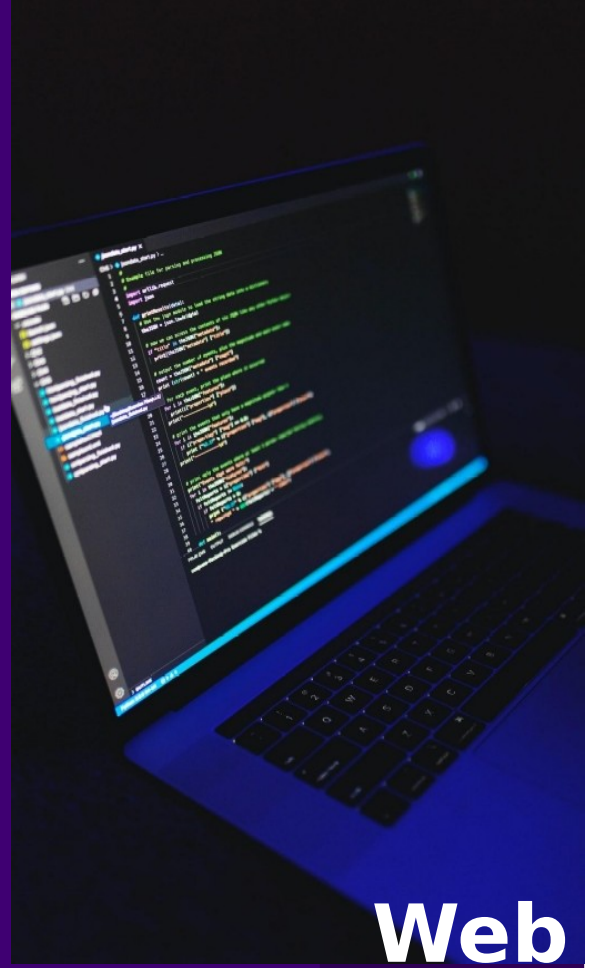
Objetivos

- **Geral**
  - Capacitar o aluno a compreender e aplicar **técnicas e ferramentas** modernas para a realização de **testes em aplicações web back-end e front-end**, enfatizando a importância dos testes no ciclo de desenvolvimento de software.
- **Específicos:**
  - Compreender o papel dos testes no processo de desenvolvimento de software e distinguir os diferentes níveis de testes.
  - Aplicar técnicas de testes no back-end, abordando testes de unidade e de integração em aplicações Spring Boot.
  - Desenvolver habilidades para conduzir testes no front-end, com foco em aplicações Angular e testes end-to-end.
  - Explorar técnicas para mensurar a cobertura de testes/código, bem como as ferramentas associadas.



Conteúdo programático

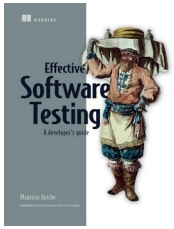
Introdução	Testes Back-end	Testes Front-end	Cobertura
O processo de Verificação, Validação e Testes (VV&T); Termos e conceitos; O que é um teste e por que testar?; Limites dos testes; Classificação de testes por nível (Taxonomia); Automatização; TDD; Frameworks.	Testes em aplicações Spring Boot; Teste de API REST; Testes de unidade (camadas de modelo, serviço e controle); Mocks; Testes de integração em controladores e repositórios de dados.	Testes em aplicações Angular; Testes de componentes; Testes de Sistema (end-to-end).	Definição; Tipos de cobertura; Cobertura de testes e cobertura de código; Nível de cobertura ideal; Ferramentas: back-end e front-end.



Web

Academy

Bibliografia



Effective Software Testing
1ª Edição – 2022
Maurício Aniche
Editora Manning

Engenharia de Software Moderna
Marco Tulio Valente
https://engsoftmoderna.info/

Introdução ao Teste de Software.
Marcio Delamaro. 2ª Edição.
Rio de Janeiro: GEN LTC, 2016.

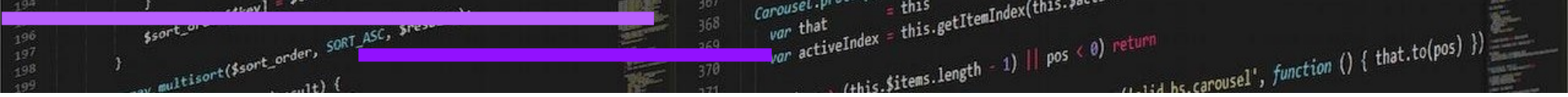
Sites de referência

- Angular Developer Guides - Testing:
  - https://angular.io/guide/
- Testing Angular - A Guide to Robust Angular Applications
  - https://testing-angular.com/
- Spring Boot Reference - Testing:
  - https://docs.spring.io/spring-boot/docs/3.0.11/reference/html/features.html#features.testing
- JUnit 5 User Guide:
  - https://junit.org/junit5/docs/current/user-guide/
- Testing Java with Visual Studio Code:
  - https://code.visualstudio.com/docs/java/java-testing

Web Academy

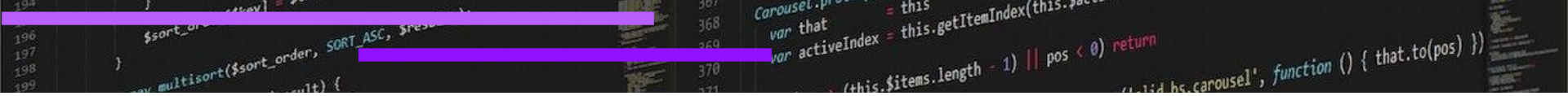
Ufac

Introdução



Introdução ao Teste de Software

- O desenvolvimento de software pode se tornar uma **tarefa bastante complexa**.
- Está sujeito a **diversos tipos de problemas** que acabam resultando na obtenção de um produto diferente daquele que se esperava.
- A maioria dos problemas tem **uma origem: o erro humano**.
- O software depende da habilidade, da interpretação e da execução das pessoas que o constroem; por isso, **erros acabam surgindo**, mesmo com a utilização de métodos e ferramentas de engenharia de software.
- **Para que os erros sejam descobertos** antes de o software ser liberado para utilização, **existe uma série de atividades, chamadas de "Verificação, Validação e Teste",** ou simplesmente **"VV&T".**



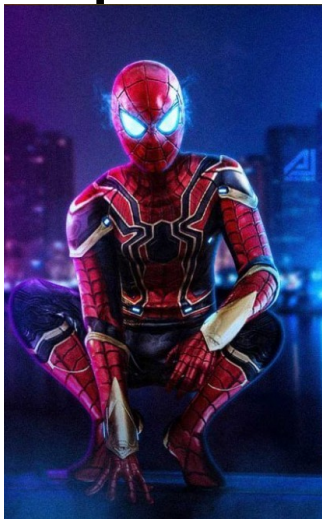
Verificação, Validação e Teste de software (VV&T)

- As atividades de **VV&T têm a finalidade de garantir** que **tanto o modo** pelo qual o software está sendo construído **quanto o produto** em si estejam em conformidade com o especificado.
- As atividades de VV&T não se restringem ao produto final. Podem e devem ser **conduzidas durante todo o processo de desenvolvimento do software**, desde a sua concepção, e englobam diferentes técnicas.
- Em geral, dividem-se as atividades de VV&T em:
  - **Estáticas** que não requerem a execução ou mesmo a existência de um programa ou modelo executável para serem conduzidas.
  - **Dinâmicas** que se baseiam na execução de um programa ou de um modelo.

Diferença entre Verificação e Validação

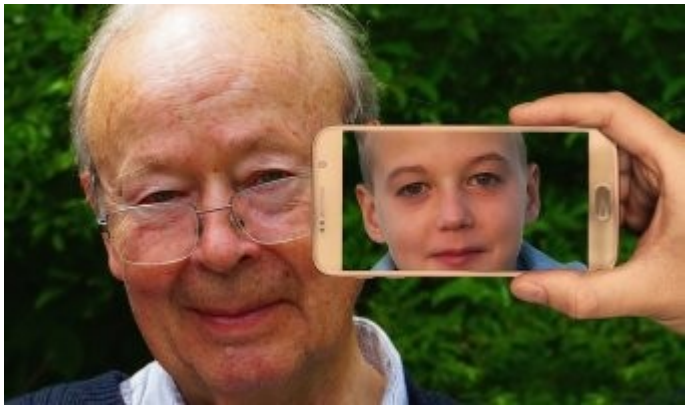
- Barry Boehm, expressou sucintamente a diferença entre validação e verificação (BOEHM, 1979).
  - **Verificação**: estamos construindo o produto da maneira certa?
  - **Validação**: estamos construindo o produto certo?

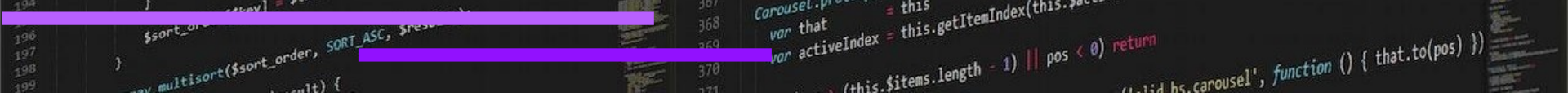
Expectativa e Realidade



Engenharia de Software: Verificação e Validação e Testes (VV&T)

- Conjunto de atividades dos processos de desenvolvimento de software.
  - **Verificação**: assegurar que o software atenda aos requisitos (funcionais e não funcionais).
  - **Validação**: assegurar que o software atenda às necessidades e expectativas do cliente.
    - **Testes** executam o código-fonte para encontrar erros, defeitos ou falhas.
    - **Inspeções** ou **revisões** (técnicas estáticas) no código ou qualquer artefato (requisitos, modelos, diagramas) sem execução para encontrar defeitos.

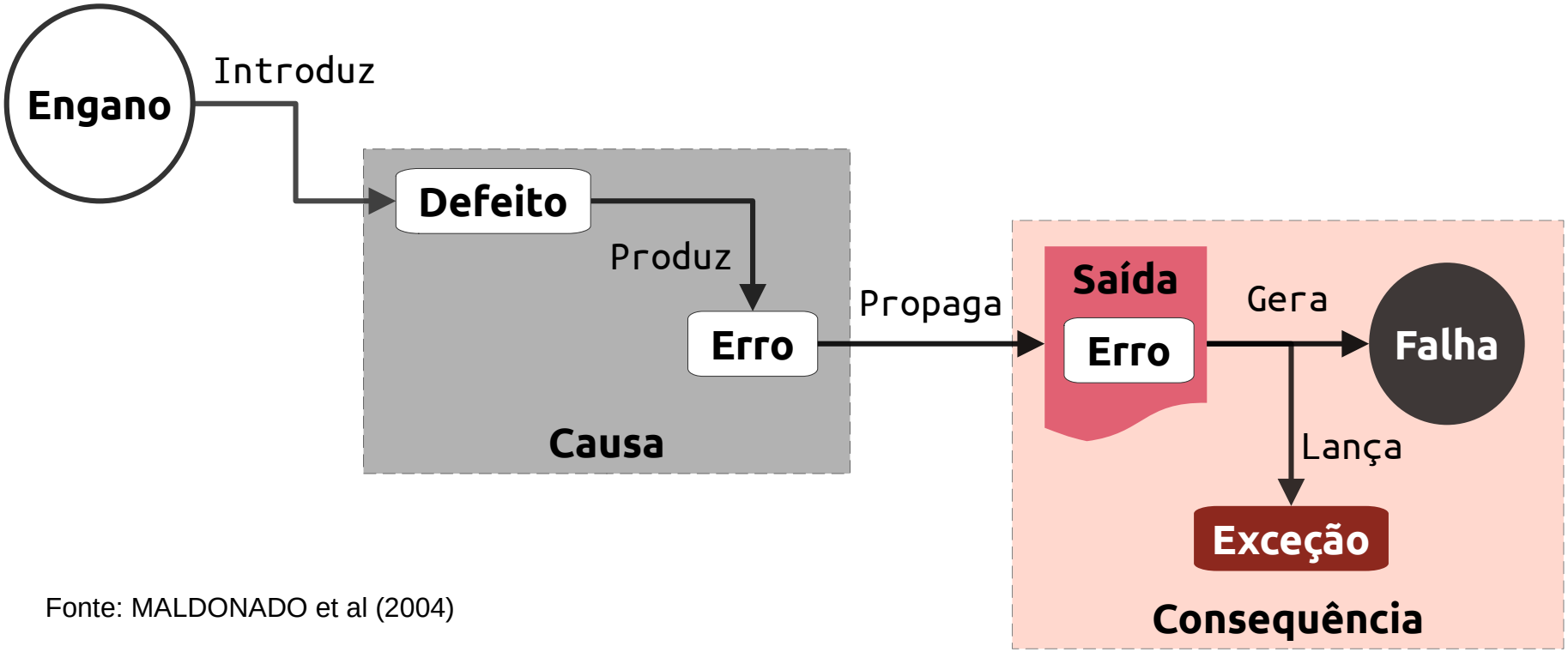




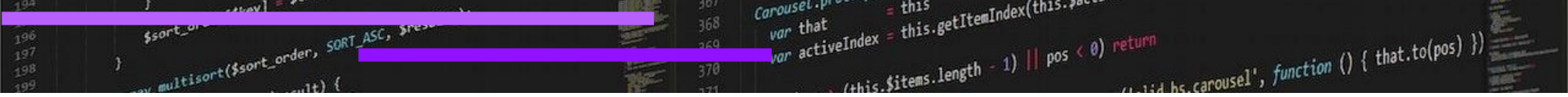
Definições de termos

- **Engano** (*mistake*): é uma ação humana que **introduz um erro**.
- **Erro** (*error*): é a **diferença entre um resultado observado** e o **resultado verdadeiro**.
- **Defeito** (*fault*)**:** é uma **manifestação de um erro**, uma imperfeição em um artefato de software que não atende a seus requisitos, precisa ser consertado e pode acarretar em uma falha.
- **Falha** (*failure*): é quando um software ou componente **executa uma função fora dos limites** especificados. Um resultado incorreto.
- **Exceção** (*exception*): é um evento que causa a **suspensão da execução normal** do software.

Engano, erro, defeito, falha e exceção

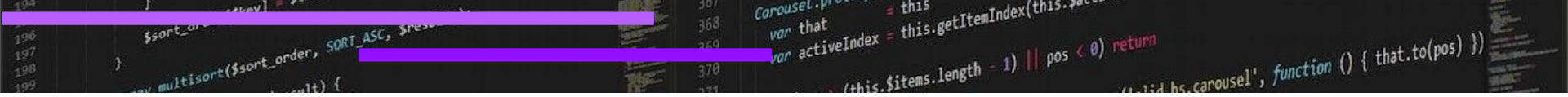


Fonte: MALDONADO et al (2004)



Por que existem falhas no software?

- Pessoas cometem erros – isso é natural/normal.
- Erros são cometidos por diversos motivos:
  - Pressão do tempo;
  - Falha humana;
  - Participantes do projeto inexperientes ou insuficientemente qualificados;
  - Falta de comunicação entre os participantes do projeto, incluindo falta de comunicação sobre os requisitos e a modelagem;
  - Complexidade do código, modelagem, arquitetura, o problema a ser resolvido ou as tecnologias utilizadas;
  - Mal-entendidos;
  - Tecnologias novas ou desconhecidas.

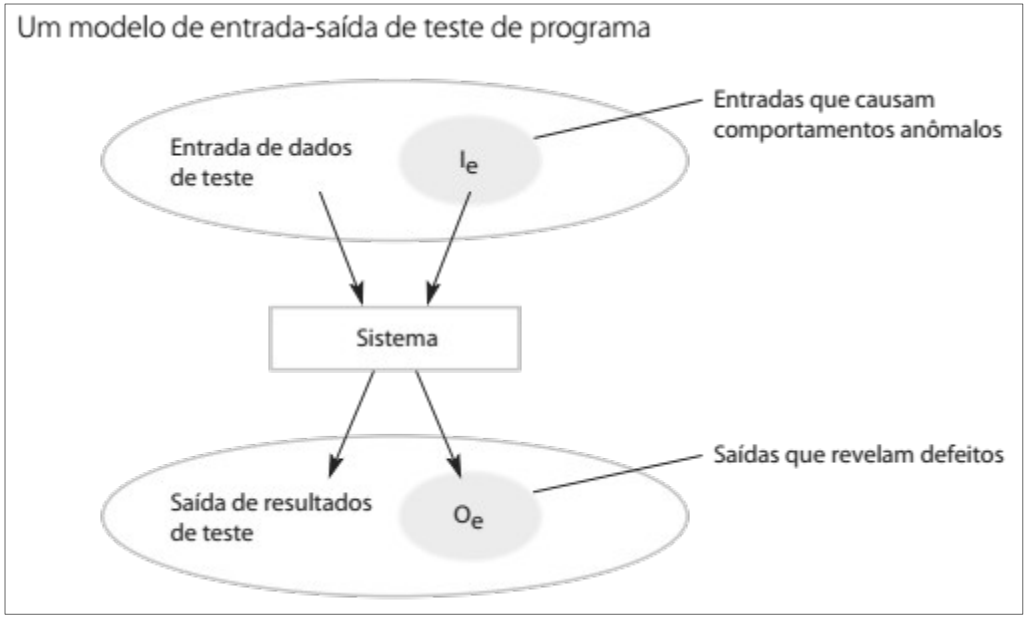


Tipos e exemplos de falhas

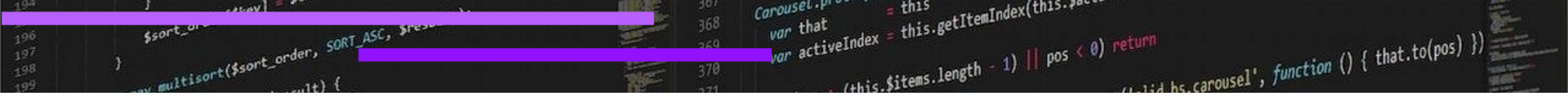
- **Visuais**
  - Problemas de alinhamento de componentes
  - Sobreposição de componentes
  - Texto impossível de ler
- **Funcionais**
  - O usuário não consegue fazer login
  - O usuário não consegue fazer pagamento com dois cartões
  - Não é possível atualizar o número de itens no carrinho de compras
  - O usuário não consegue escolher outro endereço de entrega
- **Não-funcionais**
  - Desempenho: a página demora 15 segundos para carregar
  - Segurança: a senha digitada não aparece ofuscada/oculta

O que é um teste de software?

- É um processo com objetivo mostrar que **um software faz o que é proposto a fazer**, além de descobrir os defeitos do software antes do uso.
- O teste de software consiste em executar um software com o objetivo de **revelar uma falha** (MYERS, 1979).
  - **Depuração** (*debug*) é o processo de encontrar um **defeito dado uma falha na execução do software**. Resultado de uma atividade de teste bem sucedida.

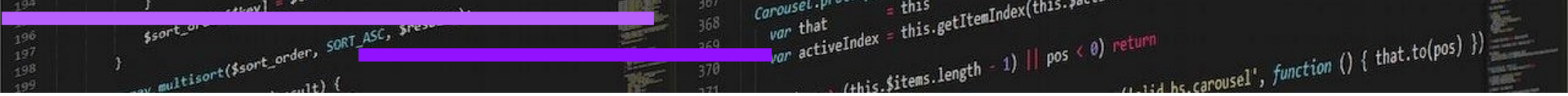


Fonte: SOMMERVILLE, 2011.



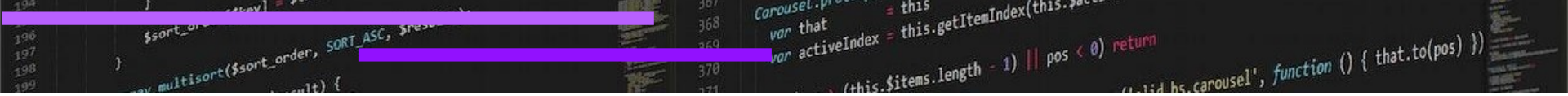
Por que testar software?

1. **Construir um produto de qualidade**: assegura que ele atende às especificações e expectativas, entregando um produto confiável ao usuário.
2. **Redução de custos**: inicialmente exige tempo e recurso, mas os testes podem reduzir gastos futuros, minimizando falhas após a implementação, o que poderia demandar mais recursos para corrigir os problemas.
3. **Eficiência no processo de desenvolvimento**: facilita a identificação e correção de erros antecipadamente, acelerando o ciclo de desenvolvimento e lançamentos.



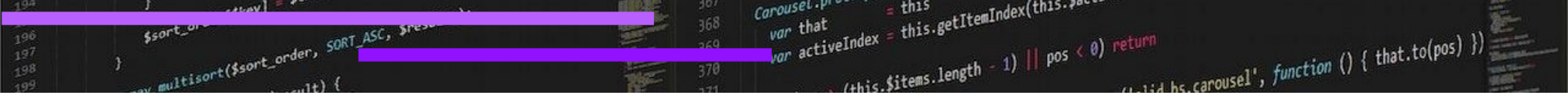
Por que testar software?

4. **Documentação**: alguém que está tentando entender um pedaço de código pode olhar para os testes para entender o que o código deve fazer.
5. **Melhorar a colaboração**: facilita a colaboração e a revisão de código entre os desenvolvedores. Ajuda a entender a intenção do código e garantir que as alterações não quebrem a funcionalidade.
6. ***Feedback* rápido**: fornece *feedback* rápido sobre a saúde do software. Isso permite que os desenvolvedores corrijam erros e *bugs* mais cedo no ciclo de desenvolvimento, o que pode economizar tempo e recursos.



Por que testar software?

7. **Satisfação do cliente**: menos *bugs* e problemas para os usuários finais resultam em uma maior satisfação do cliente.
8. **Integração contínua/Entrega contínua (CI/CD)**: permite que as alterações sejam verificadas automaticamente, facilitando a integração e entrega de alterações de código de maneira eficiente e confiável.
9. **TDD – *Test Driven Development***: metodologia que coloca os testes no centro do processo de desenvolvimento. Antes de escrever o código, os desenvolvedores primeiro escrevem um teste, então eles escrevem o código para passar o teste, e finalmente refatoram o código para padrões aceitáveis.



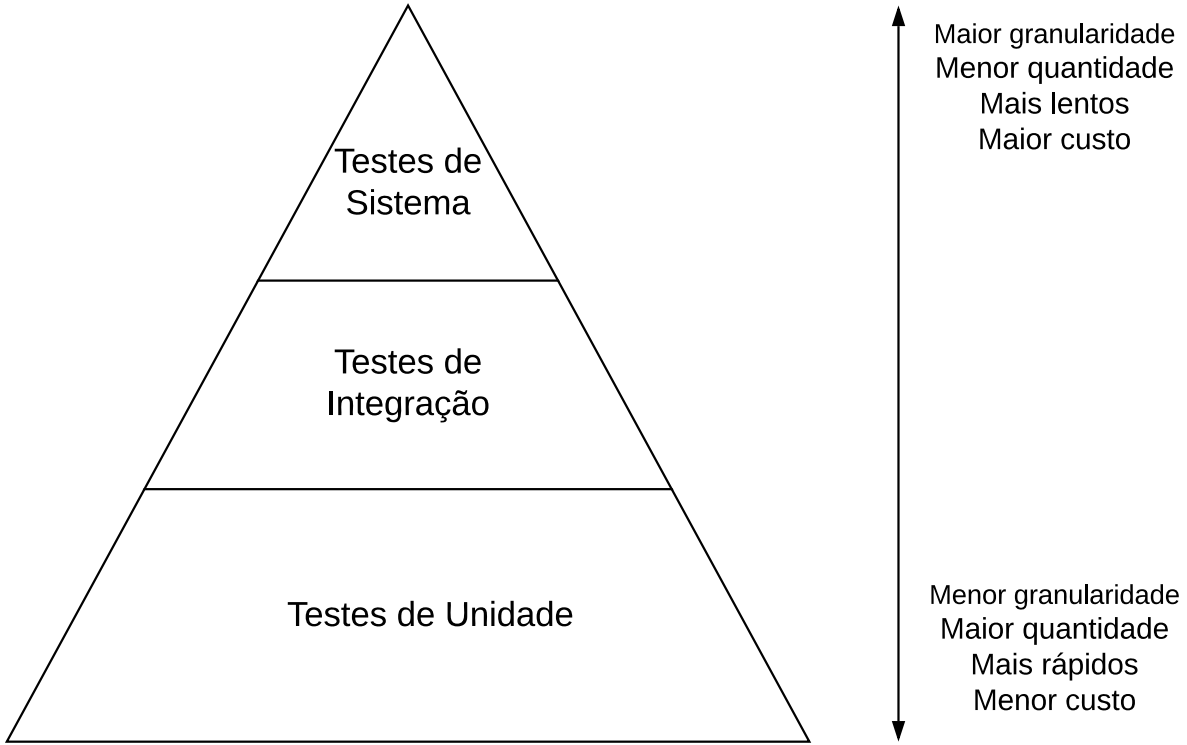
Limites dos testes

- Os testes não podem demonstrar se o software é livre de defeitos.
- É sempre possível que um teste que você tenha esquecido seja aquele que poderia descobrir mais problemas no software.

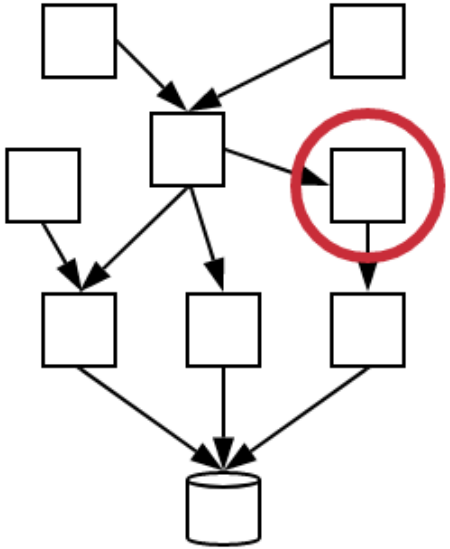
“Os testes podem mostrar apenas a presença de erros, e não sua ausência”. (Edsger Dijkstra)

Taxonomia

- Classificação de testes por nível de acordo com sua granularidade

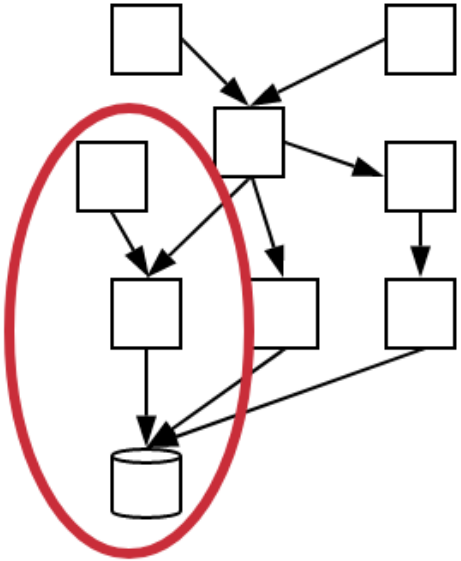


Classificação de testes por nível



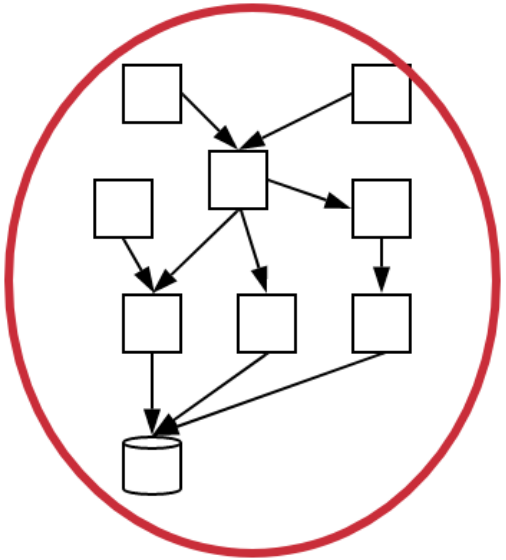
Teste de Unidade

- Verifica os componentes de forma independente (pequenas partes do código, como uma classe ou função)



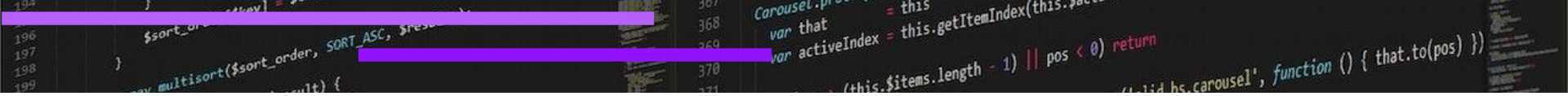
Teste de Integração

- Verificação das interações entre os componentes (internos e externos).



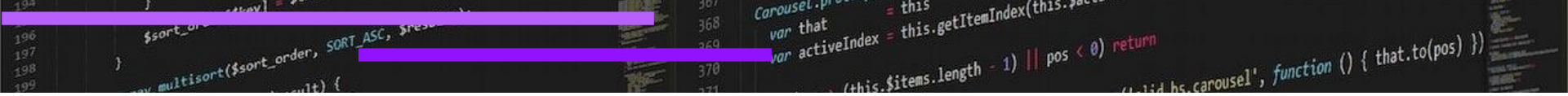
Teste de Sistema (*end-to-end*)

- Simula uma seção de uso do sistema por um usuário real



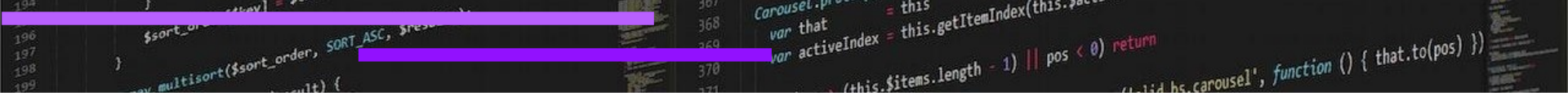
Automatização de testes

- Processo de teste geralmente envolve uma mistura de testes manuais e automatizados.
  - **Teste manual**: testador executa o programa com alguns dados de teste e compara os resultados.
  - **Teste automatizado**: testes são codificados em um programa que é executado cada vez que o software em desenvolvimento é testado.



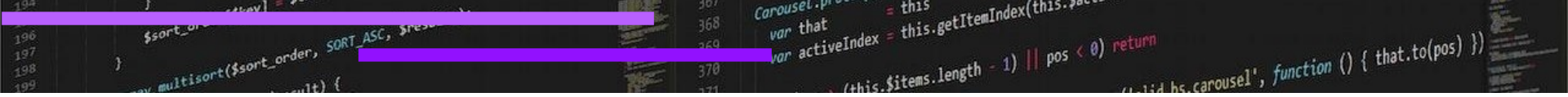
Testes de Unidade

- São **testes automatizados** de pequenas unidades de código, normalmente **classes**, as quais **são testadas de forma isolada** do restante do sistema.
- Resumidamente é um programa que **chama métodos de uma classe e verifica se eles retornam os resultados esperados**.
- Assim, quando se usa testes de unidades, o código de um sistema pode ser dividido em dois grupos: um conjunto de classes — que implementam os requisitos do sistema — e um conjunto de testes.



Testes de Integração

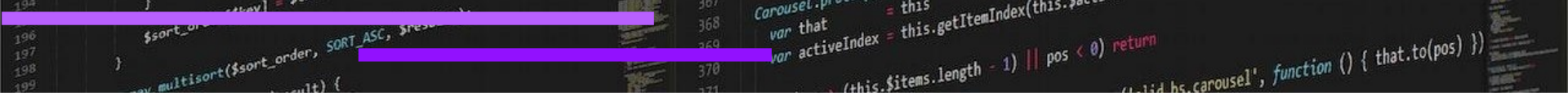
- São testes automatizados que envolvem a execução de métodos que exigem a **integração entre mais de uma unidade/camada/classe do software**.
- Na prática, o teste de integração vai ser aquele que vai testar funcionalidades que precisam **acessar o banco de dados, escrever/ler dados para/de um arquivo, invocar um serviço na rede**, etc.
- Testes de integração são **mais lentos** que testes de unidade.

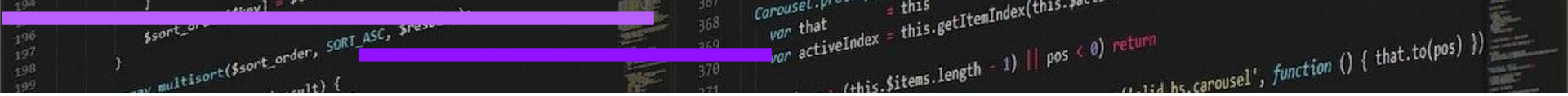


Testes de Sistema

- São testes automatizados que envolvem o teste do **software como um todo** (considerando suas operações globais, geralmente acessadas via interface gráfica ou API)
- Podem ser chamados também de **testes *end-to-end***
- Podem envolver o teste de propriedades como:
  - Desempenho
  - Segurança
  - Disponibilidade

Frameworks





Qual o defeito no método lastZero?

```
public class Example {
    // Se a == null, throw NullPointerException
    // Senão retorna o índice do último ZERO
    // Retorna -1 se não houver ocorrências de ZERO
    public static int lastZero(int[] a) {
        for(int i = 0; i < a.length; i++)
            if (a[i] == 0)
                return i;
        return -1;
    }
}
//test: a=[0,1,0] expected=2
```