Ciências da Computação

Engenharia de Software

ARA0097

Aula 05
Implementação e testes

Prof. Omar Sacilotto Donaires

Implementação e testes

- Implementação de software
- Testes de software
- Verificação e validação
- Estratégia de teste de software
- Estratégias de teste para software convencional

Objetivos

Saber o quê	Saber como	Saber porque
Compreender a importância de testes de software, como forma de aferir a qualidade do produto gerado.		Para ser capaz de produzir softwares com qualidade (sem erros).

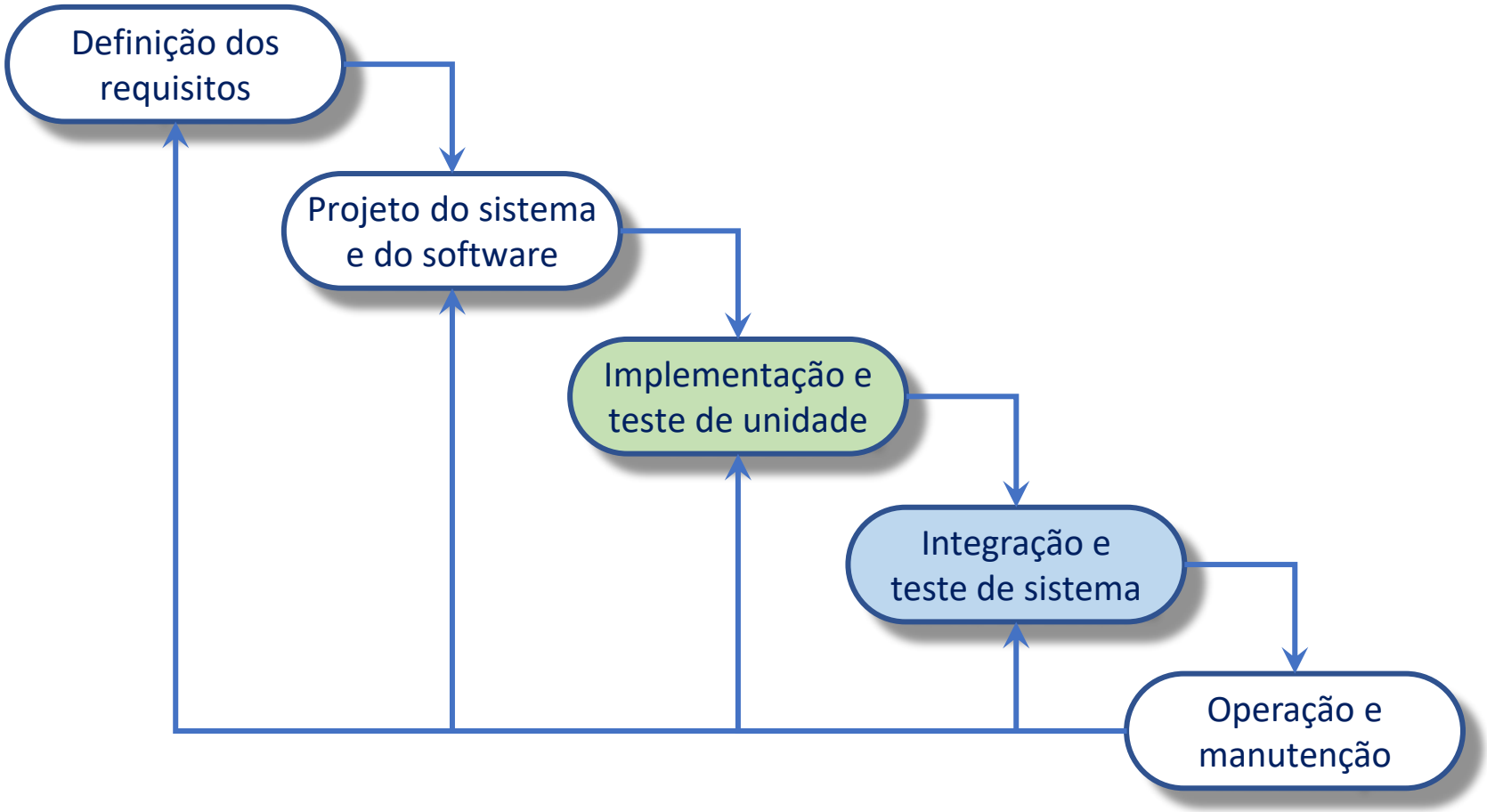
Estácio

Situação-problema

- Imagine que estamos desenvolvendo um **software** responsável por **realizar cirurgias** com **robôs** e iremos **colocá-lo em produção** em breve.
- Como podemos **garantir** que esse sistema é **seguro** e **não** irá apresentar **falhas**?

Modelo em cascata

Sommerville, 2011



Estácio

Implementação de software

Implementação

Sommerville, 2006, p. 47

- Processo de **conversão** da **especificação de sistema** em um **sistema executável**.

- Envolve:
  - atividades de projeto
  - programação de software
  - aperfeiçoamento da especificação de software, no caso de uma abordagem evolucionária de desenvolvimento

Codificação

Mazzola, 2010, p. 103, 109

- Atividades relacionadas à **utilização de uma linguagem de programação**
  - traz maior proximidade com a linguagem processada pela máquina.
- **Código-fonte** → elemento **essencial**
  - para as atividades de **validação** do software
  - para as tarefas de **manutenção**
- O aspecto de **documentação** deve cuidadosamente considerado na etapa de codificação

Linguagens de programação para garantir emprego SEMPRE

- Java
- PHP
- CSS
- C
- C++
- C#
- JavaSccript
- Kotlin
- Swift
- Go

- Artigo original → 2015
- Da lista original para cá
  - Perderam um pouco da notoriedade:
    - Perl, Objetive-C, Visual Basic, Ruby e R.
  - Continuam muito relevantes:
    - Java, JavaScript, CSS, PHP, Python e C
  - Ganharam notoriedade:
    - C++, C#, Kotlin, Swift e Go.

https://targettrust.com.br/blog/linguagens-de-programacao-garantia-emprego/

Estácio

Principais SGBDs

- **Oracle Database** – mais utilizado no mundo.
- **SQL Server** – criado pela Microsoft em 1988.
- **MySQL** – Open Source, ou seja, de código aberto.
- **PostgreSQL** – BD relacional e Open Source.
- **DB2** – da gigante IBM.
- **MongoDB** – sistema NoSQL (não relacional), Open Souce.
- **Redis** – BD NoSQL baseado em chave-valor.
- **InfluxDB** – BD de séries temporais.
- **DynamoDB** – BD NoSQL As A Service, criado pela Amazon.
- **Cassandra** – BD NoSQL, atualmente mantido pela fundação Apache.
- **Microsoft Access** – SGBD relacional, integrante do pacote Office.
- **SQLite** – biblioteca em linguagem C, que permite usar um BD SQL.
- **MariaDB** – igual ao MySQL, forte preocupação com segurança.

https://www.opservices.com.br/banco-de-dados/

https://becode.com.br/principais-sgbds/

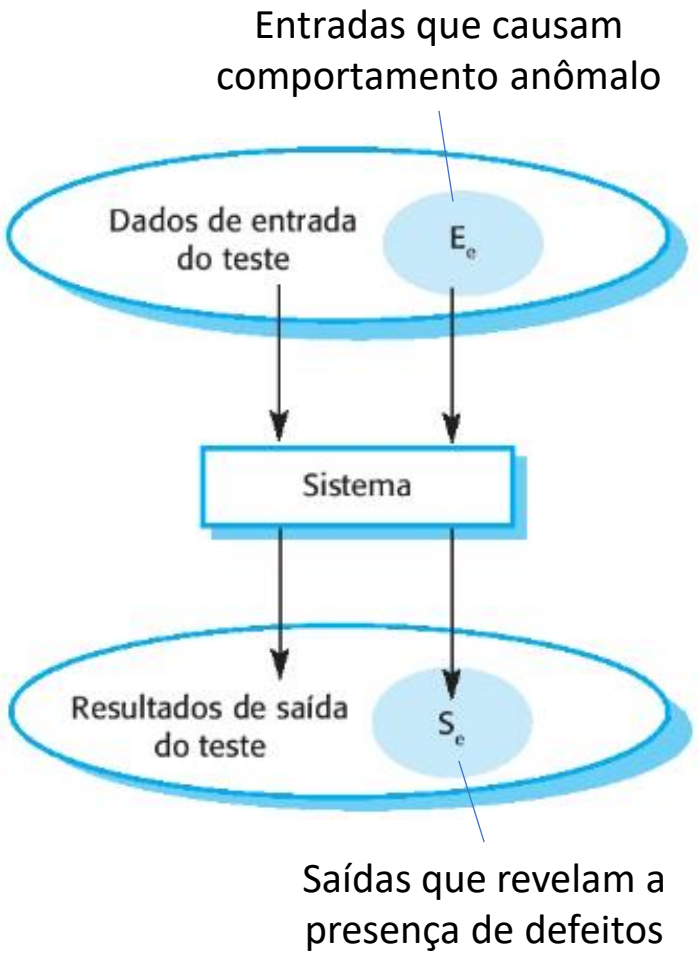
Estácio

Testes de software

Testes

Sommerville, 2011

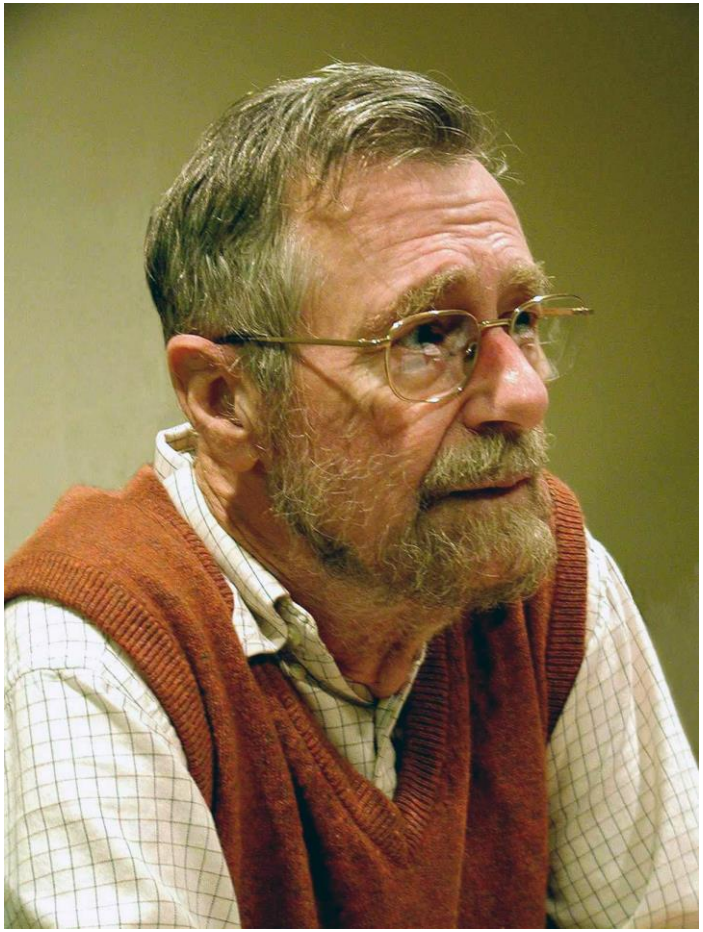
- **Execução** de um programa com **dados artificiais** e **análise** dos **resultados** em busca de **erros**, **anomalias** ou informações sobre os atributos não funcionais do programa.
- Objetivos:
  - Demonstrar que o software **atende** aos seus **requisitos** → **Teste de validação**
  - Descobrir **comportamento incorreto**, indesejável ou fora da conformidade com suas especificações → **Teste de defeitos**



Estácio

Teste

Edsger Dijkstra (1972)



"O teste só consegue mostrar a presença de erros, não a sua ausência."

Estácio

Teste de software

Pressman e Maxim, 2016

- **Não produz a qualidade** →
  - Se a qualidade não está lá antes de um teste, ela não estará lá quando o teste terminar.
  - A qualidade é incorporada ao software por meio do processo de engenharia de software.
- Elemento de um **tema mais amplo**, conhecido como **verificação e validação** (**V&V**).
- **Último elemento** a partir do qual
  - a **qualidade** pode ser **estimada** e
  - os **erros** podem ser **descobertos**.

Estácio

Concepções a respeito de teste

Pressman e Maxim, 2016

- É correto afirmar que:
  - Os **erros** estão **presentes** → se o engenheiro de software não encontrar os erros, o cliente encontrará!
  - Os **interesses** dos **desenvolvedores** vão contra o teste completo → os desenvolvedores têm interesse em demonstrar que o programa :
    - é **isento de erros**
    - **funciona** de acordo com os **requisitos** do cliente
    - será concluído dentro do **prazo** e do **orçamento** previstos.
- Noções incorretas:
  - O desenvolvedor de software não deve fazer nenhum teste
  - O software deve ser "atirado aos leões" → entregue a estranhos que realizarão testes implacáveis.
  - Os testadores se envolvem no projeto somente no início das etapas do teste.

Estácio

Grupo independente de teste
Independent test group (ITG)
Pressman e Maxim, 2016

- O **desenvolvedor** é responsável
  - Pelo teste das **unidades** individuais (componentes).
  - Em muitos casos, também pelo teste de **integração**.
- **Grupo de teste independente** (ITG)
  - Envolve-se depois que a **arquitetura** do software está **concluída**.
  - **Remove** o **conflito de interesses**.
  - Pago para **encontrar erros**.
  - Em muitos casos o ITG se reporta à organização de garantia da qualidade do software.
- O **desenvolvedor** e o **pessoal do ITG** trabalham **juntos** durante todo o projeto de software
  - Para garantir que testes completos sejam realizados.
  - Enquanto o teste está sendo realizado, o desenvolvedor deve estar disponível para corrigir os erros encontrados.
  - O ITG faz parte da equipe de desenvolvimento de software, pois se envolve durante a análise e o projeto.


Testes de
Software

Verificação e validação

Verificação e validação
Boehm, 1981

- Verificação
  - Conjunto de tarefas que garantem que o software **implementa corretamente** uma **função** especifica.
  - "Estamos **criando o produto corretamente**?"
- Validação
  - Conjunto de tarefas que **asseguram** que o software foi criado e pode ser rastreado segundo os **requisitos do cliente**.
  - "Estamos **criando o produto certo**?"

Abrangem muitas **atividades** de **garantia da qualidade do software** (software quality assurance, SQA).

Estácio

Inspeções e revisões
Sommerville, 2011

- **Técnicas estáticas** de V&V → não é necessário executar o software para verificá-lo.
- Visam **analisar e conferir**
  - os **requisitos** do sistema
  - os **modelos** de projeto
  - o **código-fonte d**o programa
  - até mesmo os **testes** de sistema propostos.
- Possibilitam verificar
  - Conformidade com padrões
  - Ineficiências
  - Algoritmos inadequados
  - Mau estilo de programação...
- Não podem substituir o teste de software
  - Não são boas para descobrir defeitos resultantes de interações, problemas de temporização ou de desempenho.

Concentram-se principalmente no código fonte.

Estácio

Estratégia de teste de software

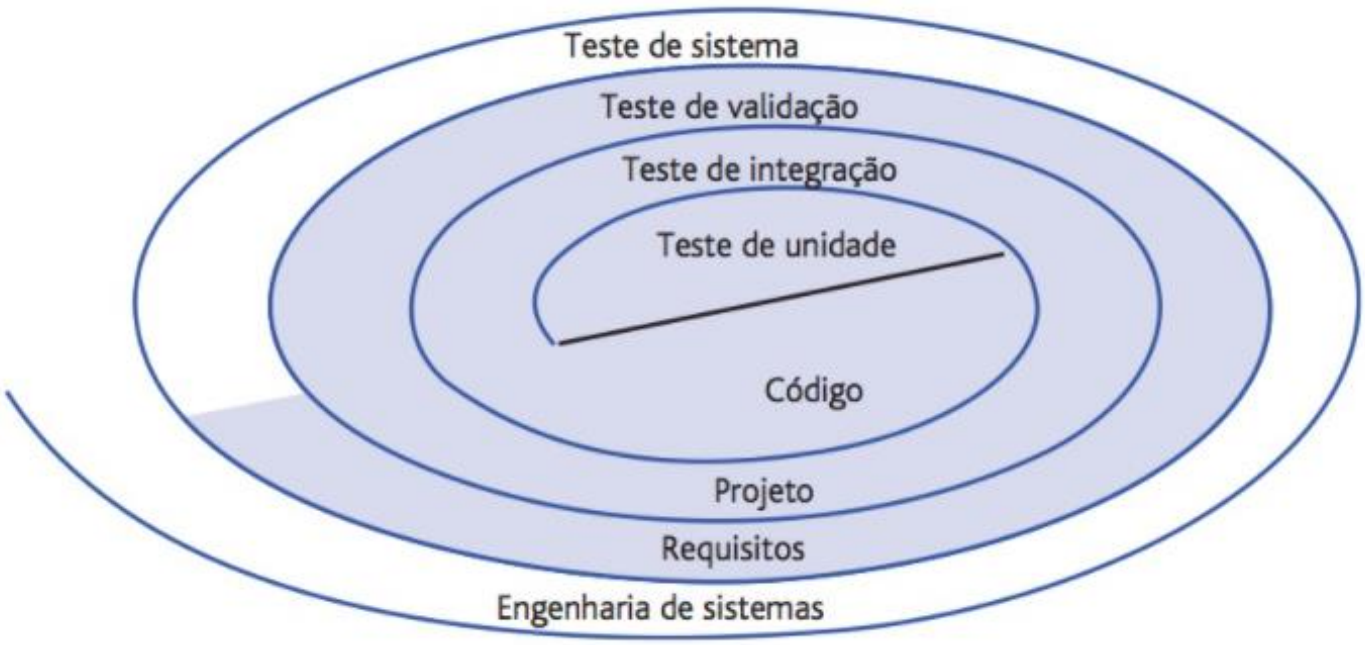
Estratégia de teste de software

Pressman e Maxim, 2016

- Fornece um **roteiro** que descreve os **passos** a serem executados como parte do **teste**.
  - Define quando esses passos são planejados e executados.
  - Define quanto trabalho, tempo e recursos serão necessários.
- **Teste** é um conjunto de **atividades** que podem ser
  - **planejadas** com antecedência e
  - **executadas sistematicamente**.
- Deve incorporar:
  - **Planejamento** dos testes
  - Projeto de **casos de teste**
  - **Execução** dos testes
  - **Coleta** e **avaliação dos dados** resultantes.

Estratégia de teste de software vista no conceito de uma espiral

Pressman e Maxim, 2016

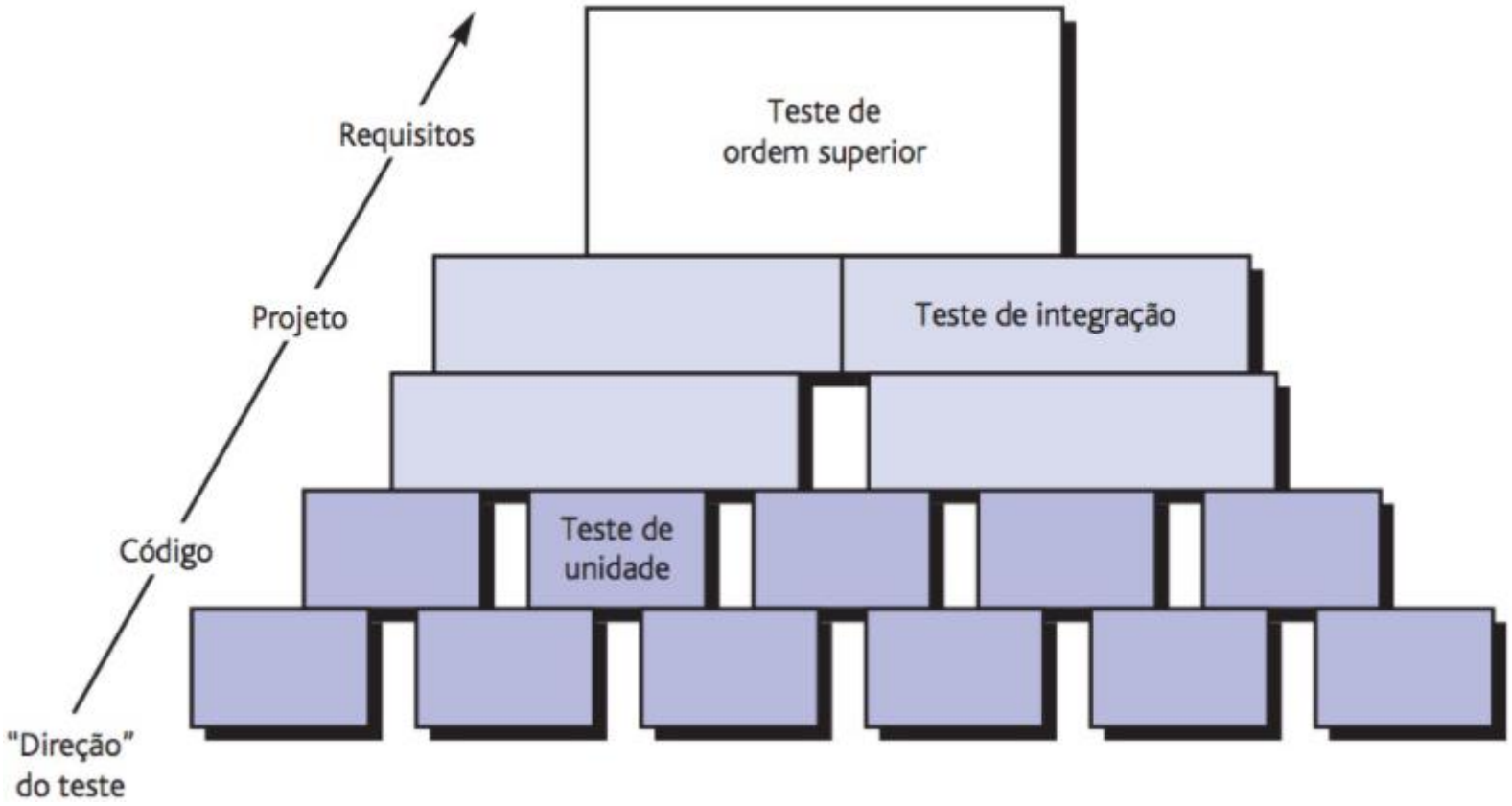


- Para **desenvolver** software de computador, percorre-se a espiral **para o interior** → ao longo de linhas que indicam a diminuição do nível de abstração a cada volta.
- Para **testar** um software de computador, percorre-se a espiral **para o exterior** → ao longo de linhas que indicam o escopo do teste a cada volta.

Estácio

Etapas de teste de software

Pressman e Maxim, 2016



Estácio

Etapas do processo de teste de software
Pressman e Maxim, 2016

- Teste de **unidade**
  - Focalizam cada **componente**, usando técnicas que exploram **caminhos** específicos na **estrutura de controle** de um componente.
- Teste de **integração**
  - Depois que os componentes são integrados no **pacote completo**.
  - Focalizam **entradas e saídas** são mais predominantes.
- Teste de **validação**
  - Proporciona a **garantia** final de que o software **satisfaz** a todos os **requisitos** funcionais, comportamentais e de desempenho.
- Teste de **sistema**
  - Extrapola os limites da engenharia de software → entra no contexto da engenharia de sistemas de computadores.
  - Testa o software em combinação com **outros elementos do sistema** (por exemplo, hardware, pessoas, base de dados).

Estácio

Critérios para conclusão do teste
Pressman e Maxim, 2016

- Pergunta clássica
  - "**Quando** podemos dizer que **terminamos os testes**?"
- Respostas pragmáticas:
  - "O teste **nunca** termina; o encargo simplesmente passa do engenheiro de software para o usuário".
  - "O teste acaba **quando o tempo ou o dinheiro acabam**" (resposta um tanto cínica, mas exata).
- Abordagem engenharia de software **sala limpa**
  - **Critério mais rigoroso**
  - Coletando **métricas** durante o teste do software e utilizando **modelos estatísticos** é possível desenvolver diretrizes significativas para responder à pergunta original.

Estratégias de teste para software convencional

Teste de unidade

Pressman e Maxim, 2016

- Focaliza a menor unidade de projeto do software – o **componente** ou módulo de software.
  - Testa **caminhos de controle** importantes para descobrir erros dentro dos limites do módulo.
  - Enfoca a **lógica interna** de processamento e as estruturas de dados dentro dos limites de um componente.
- Testa
  - Interface do módulo
  - Estrutura de dados locais
  - Todos os caminhos independentes da estrutura de controle
  - Condições limite
  - Caminhos de manipulação de erro

Abordagens de integração

Pressman e Maxim, 2016

- Integração “**big bang**”
  - Todos os componentes são combinados antecipadamente.
  - O programa inteiro é testado como um todo.
  - Normalmente, o resultado é o caos!
- Integração **incremental**
  - O programa é **construído** e **testado** em pequenos **incrementos**.
  - Os erros são mais fáceis de isolar e corrigir; as interfaces têm maior probabilidade de ser testadas completamente;
  - Uma abordagem sistemática de teste pode ser aplicada.

Estratégias de integração incremental

Pressman e Maxim, 2016

- Integração **descendente**
  - **Começa** com o **módulo** de controle **principal**
  - Desloca-se **para baixo** pela hierarquia de controle.
    - **Módulos subordinados** ao módulo de controle principal são incorporados à estrutura de uma maneira primeiro-em-profundidade ou primeiro-em-largura.
- Integração **ascendente**
  - **Começa** a construção e o teste com **módulos atômicos**.
  - Os componentes são integrados **de baixo para cima** → elimina a necessidade de pseudocontroladores.

Testes de integração

Pressman e Maxim, 2016

- Teste de **regressão**
  - **Reexecução** do **mesmo subconjunto de testes** que já foram executados cada vez que um novo módulo é acrescentado como parte do teste de integração.
  - O objetivo é assegurar que as **alterações** não tenham propagado **efeitos colaterais** indesejados.
- Teste **fumaça**
  - **Realizados frequentemente** (diariamente).
  - Não precisa ser exaustivos mas deve **exercitar** o **software como um todo** (de cabo a rabo).
  - O objetivo é descobrir problemas que têm grande probabilidade de atrasar o cronograma.
  - **Minimiza** o **risco de integração**.

Artefatos de integração

Pressman e Maxim, 2016

- **Especificação de Teste**
  - Incorpora um **plano de teste**
    - Incluindo um cronograma dos testes.
  - Descreve o **procedimento de teste** detalhado
    - Incluindo uma lista de todos os casos de teste
    - e dos resultados esperados.
- **Relatório de Teste**
  - Registra um histórico dos **resultados reais** do teste, problemas ou peculiaridades.
  - Essas informações podem ser vitais durante a **manutenção** do software.

Estratégias de Teste de Software: O Case WhatsApp

https://www.youtube.com/watch?v=id8Yf3iXroY

Estácio

Atividade acadêmica avaliativa

- **Vale 1,0 ponto**, para compor a nota da **AV1**.
- ATIVIDADE **EM GRUPOS** DE **ATÉ 3 PESSOAS**.
- Escolher um sistema que seja do conhecimento de todos (rede social, aplicativo de mensagens, delivery, internet banking, etc.)
- Listar 10 casos de teste, que podem abordar diversos cenários.
- Apresentar os seus casos de teste, para os demais alunos.

Estácio

Leitura específica

- Vídeo "Estratégias de Teste de Software: O Case WhatsApp".
  - Disponível em:
  - https://www.youtube.com/watch?v=id8Yf3iXroY
- PRESSMAN, Roger; MAXIM, Bruce. Engenharia de Software. Porto Alegre: AGMH, 2016.
  - Disponível em:
  - https://integrada.minhabiblioteca.com.br/#/books/9788580555349/
  - Capítulo 22, seções 22.1 a 22.3 (Páginas 466 a 480)

Estácio

Aprenda mais

- SANTOS, Glyciane. Melhores práticas na elaboração de casos de teste.
  - Disponível em:
  - https://blog.cedrotech.com/melhores-praticas-na-elaboracao-de-casos-de-teste/

- FERNANDES, Mateus. Teste de Unidade e Teste de Integração: O que são?
  - Disponível em:
  - https://medium.com/@mateus1198/teste-de-unidade-e-teste-de-integração-o-quesão-de58d7a3d3d2

Estácio

Para próxima aula

- Vídeo “Você ainda faz testes de software manuais?”
  - Disponível em:
  - https://www.youtube.com/watch?v=fxL04i00jJw

- SOMMERVILLE, Ian. Engenharia de Software. 10ª Ed. São Paulo: Pearson Prentice Hall, 2011.
  - Disponível em:
  - https://plataforma.bvirtual.com.br/Leitor/Loader/168127/pdf
  - Capítulo 8, seções 8.1 a 8.4 (Páginas 203 até 230)

Estácio

Mãos à obra

Estácio

Nesta aula aprendemos:

- Aspectos da implementação de software
- A diferença entre verificação e validação
- Estratégias de teste de software.
- Estratégias de integração.