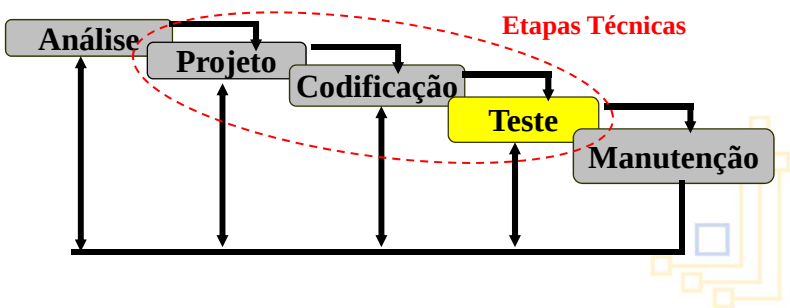
ENGENHARIA DE SOFTWARE

UNIDADE 6 – Técnicas de Testes de Software
(Aula 9 – Tipos de Estratégias)

Prof. Ivan Nunes da Silva

Função do Teste de Software

- **Função da Etapa de Teste:**
  - **O objetivo principal da atividade de teste é revelar erros ainda não descobertos.**



2

Finalidades do Teste de Software

- **Testar é um processo de executar um programa com a intenção de encontrar erros.**
- **Um bom teste é aquele com alta probabilidade de encontrar erros ainda não descobertos.**
- **Um teste bem sucedido é o que encontra erros.**

3

Estratégia de Teste de Software

- A atividade de teste de software é um elemento de um tema mais amplo chamado ***Verificação*** *e* ***Validação*** (V&V).
  - **VERIFICAÇÃO → refere-se ao conjunto de atividades que garante que o software implemente corretamente uma função específica.**
  - **VALIDAÇÃO → refere-se ao conjunto de atividades que garante que o software que foi construído está de acordo com as exigências do cliente.**
- A definição de V&V abrange muitas das atividades às quais nos referimos como **Garantia da Qualidade de Software** (SQA).

4

Organização do Teste de Software

- Quando o teste se inicia há um conflito de interesses:
  - **Desenvolvedores** → têm interesse de demonstrar que o programa é isento de erros.
  - **Os responsáveis pelos testes** → têm interesse em mostrar que o programa tem erros.
- Do ponto de vista **psicológico**:
  - Análise, Projeto e Codificação de Software são tarefas **construtivas**.
  - Teste é tarefa **destrutiva**.
- Para suprir o conflito de interesses existe o **Grupo Independente de Testes** (ITG):
  - O ITG faz parte da equipe de projeto de desenvolvimento de software, no sentido de que está envolvido durante o processo de especificação e continua envolvido planejando e especificando procedimentos de teste ao longo de um grande projeto.

5

Estratégias de Teste de Software

- **Teste de Unidade**
  - Concentra-se em cada unidade do software, de acordo com o que é implementado no código fonte.
  - Cada módulo é testado individualmente garantindo que ele funcione adequadamente.
- **Teste de Integração**
  - Concentra-se no projeto e na construção da arquitetura de software.
  - Os módulos são montados ou integrados para formarem um pacote de software.
- **Teste de Validação**
  - Os requisitos estabelecidos com a parte da análise de requisitos de software são validados em relação ao software que foi construído.
- **Teste de Sistema**
  - O software e outros elementos do sistema são testados como um todo.

6

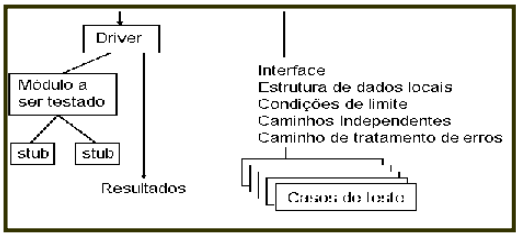
O Teste de Unidade

- **Avaliação de Interface**
  - A ***interface*** com o módulo é testada para ter a garantia de que as informações fluem para dentro e para fora da unidade de programa que se encontra sob teste.
- **Avaliação da Estrutura de Dados**
  - A ***estrutura de dados*** local é examinada para ter a garantia de que dados armazenados temporariamente mantêm sua integridade durante todos os passos de execução.
- **Avaliação das Condições Limites**
  - As ***condições de limite*** são testadas para ter a garantia de que o módulo opera adequadamente nos limites estabelecidos para demarcarem ou restringirem o processamento.
- **Avaliação de Caminhos Independentes**
  - Todos os ***caminhos independentes*** através da estrutura de controle são exercitados para ter a garantia de que todas as instruções de um módulo foram executadas pelo menos uma vez.

7

O Teste de Unidade (Ambiente)

- Uma vez que o módulo não é um programa individual, um software *driver* e/ou *stub* deve ser desenvolvido para cada unidade de teste.
  - **Driver** → Na maioria das aplicações é um programa principal que aceita dados de caso de teste, passa tais dados para o módulo a ser testado e imprime os dados relevantes.
  - **STUB ou Programa Simulado** → servem para substituir módulos que estejam subordinados (chamados pelo) ao módulo a ser testado. Usa a interface do módulo subordinado, pode fazer manipulação de dados mínima, imprime verificação de entrada e retorna.



8

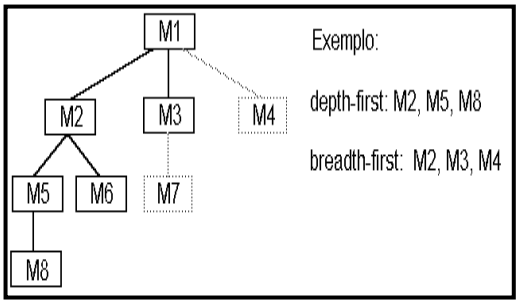
O Teste de Integração

- É uma técnica sistemática para a construção da estrutura de programa, realizando-se, ao mesmo tempo, testes para descobrir erros associados a interfaces.
- O objetivo é, a partir dos módulos testados ao nível de unidade, construir a estrutura de programa que foi determinada pelo projeto.
- ***Integração Não Incremental***
  - Abordagem "big-bang".
  - O programa completo é testado como um todo e o resultado é o caos.
  - Quando erros são encontrados, a correção é difícil porque o isolamento das causas é complicado pela vasta amplitude do programa inteiro.
- ***Integração Incremental***
  - O programa é construído e testado em pequenos segmentos, onde os erros são mais fáceis de serem isolados e corrigidos.
  - As interfaces têm maior probabilidade de serem testadas completamente e uma abordagem sistemática ao teste pode ser aplicada.

9

O Teste de Integração (Top-Down)

- Os módulos são integrados movimentando-se de cima para baixo, através da hierarquia de controle, iniciando-se do módulo de controle principal.
- Os módulos subordinados ao módulo de controle principal são incorporados à estrutura de uma maneira ***depth-first*** (primeiro pela profundidade) ou ***breadth-first*** (primeiramente pela largura).



10

O Teste de Integração (Top-Down)

- **Passos do Processo de Integração (Top-Down)**
  1. O módulo de controle é usado como um *driver* de teste e os *stubs* são substituídos para todos os módulos diretamente subordinados ao módulo de controle principal.
  2. Dependendo da abordagem de integração escolhida, os *stubs* subordinados são substituídos, um de cada vez por módulos reais.
  3. Testes são realizados à medida que cada módulo é integrado.
  4. Durante a conclusão de cada conjunto de testes, outro *stub* é substituído pelo módulo real.
  5. Teste de regressão - (isto é, a realização de todos ou de alguns dos testes anteriores) pode ser realizado a fim de garantir que novos erros não tenham sido introduzidos.

11

O Teste de Validação

- A **Especificação de Requisitos de Software** contém os critérios de validação que formam a base para uma abordagem ao teste de validação.
- A validação do software é realizada por meio de uma série de testes que demonstram a conformidade com os requisitos.
- O plano e o procedimento de teste são projetados para garantir que:
  - *Todos os requisitos funcionais são satisfeitos.*
  - *Todos os requisitos de desempenho são conseguidos.*
  - *A documentação está correta.*
  - *Outros requisitos são cumpridos: portabilidade, compatibilidade, remoção de erros e manutenibilidade.*

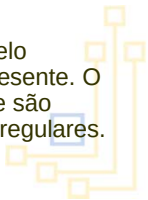
12

O Teste de Validação (Alfa & Beta)

- É virtualmente impossível que se preveja como o cliente ***realmente*** usará um programa.
- As instruções de uso podem ser mal interpretadas.
- Combinações estranhas de dados podem ser regularmente usadas.
- Saídas que pareciam claras ao analista podem ser ininteligíveis para um usuário em campo.

O Teste de Validação (Alfa & Beta)

- **Software Customizado para um Cliente**
  - São realizados ***testes de aceitação***, realizados pelo usuário final para capacitá-lo e para validar todos os requisitos.
  - Pode variar de um *test-drive* informal a uma série de testes planejados.
- **Software Desenvolvido como um Produto para Muitos Clientes**
  - **Teste Alfa** → É levado a efeito por um cliente nas instalações do desenvolvedor. O software é usado num ambiente controlado com o desenvolvedor analisando o usuário e registrando erros e problemas de uso.
  - **Teste Beta** → É realizado nas instalações do cliente pelo usuário final do software. O desenvolvedor não está presente. O cliente registra os problemas (reais ou imaginários) que são encontrados e relata-os ao desenvolvedor a intervalos regulares.



O Teste de Sistema

- O software é apenas um elemento de um sistema baseado em computador mais amplo.
- O teste de sistema envolve uma série de diferentes testes, cujo propósito primordial é por completamente à prova o sistema baseado em computador.
- Problema Clássico da atividade de teste de sistema:
  - Quando ocorre um erro, o desenvolvedor de um elemento do sistema culpa outro pelo problema.

O Teste de Sistema

- O teste de sistema envolve uma série de diferentes testes, cujo propósito primordial é por completamente à prova o sistema baseado em computador.
- O engenheiro de software deve antecipar os problemas potenciais da interface:
  1. Projetar caminhos de tratamento de erros que testem todas as informações que chegam de outros elementos do sistema.
  2. Realizar uma série de testes que simulem dados ruins ou outros erros em potencial na interface do software.
  3. Registrar os resultados de teste para usar como "prova" se alguém lhe responsabilizar por eventuais erros.
  4. Participar do planejamento e projeto dos testes do sistema para garantir que o software seja adequadamente testado.

O Teste de Sistema (Teste de Recuperação)

- Muitos sistemas baseados em computador precisam recuperar-se de falhas e retomar o processamento dentro de um tempo previamente especificado.
- Em certos casos, o sistema deve tolerar falhas, isto é, o processamento de falhas não deve fazer com que uma função global de sistema cesse.
- Em outros casos, uma falha do sistema deve ser corrigida dentro de um período previamente especificado; caso contrário, graves prejuízos econômicos ocorrerão.
- O **Teste de Recuperação** é um teste de sistema que força o software a falhar de diversas maneiras e verifica se a recuperação é adequadamente executada.

17

O Teste de Sistema (Teste de Segurança)

- O **Teste de Segurança** tenta verificar se todos os mecanismos de proteção embutidos num sistema o protegerão, de fato, de acessos indevidos.
- Durante o teste de segurança, o analista desempenha o papel de pessoa que deseja penetrar no sistema. Algumas práticas são:
  - Adquirir senhas mediante contatos externos.
  - Atacar o sistema com software customizado projetado para derrubar quaisquer defesas que tenham sido construídas.
  - Desarmar o sistema, negando serviços a outros.
  - Provocar erros intencionalmente, esperando acessá-lo durante a recuperação.
  - Mitigar através de dados inseguros esperando descobrir a chave para a entrada no sistema.

18

A Arte da Depuração (Processo)

- O objetivo primordial da **Depuração** é descobrir e corrigir a causa de um erro de software.
- A **Depuração** ocorre em consequência de testes bem sucedidos.
- Embora a depuração possa e deva ser um processo disciplinado, ela ainda tem muito de arte (experiência).
- Durante a depuração, encontram-se erros que variam de brandamente perturbadores a catastróficos.
- À medida que as consequências de um erro aumenta, a pressão para descobrir a causa também aumenta.
- **POR QUE A DEPURAÇÃO É TÃO DIFÍCIL?**
  1. *O sintoma e a causa podem ser geograficamente remotos. Estruturas altamente acopladas agravam essa situação.*
  2. *O sintoma pode desaparecer (temporariamente) quando outro erro é corrigido.*
  3. *O sintoma pode, de fato, ser causado por não erros (por exemplo, imprecisões de arredondamento).*
  4. *O sintoma pode ser causado por um erro humano que não é facilmente rastreado.*
  5. *O sintoma pode ser resultado de problemas de timing, e não problemas de processamento.*
  6. *O sintoma pode ser devido a causas que estão distribuídas por uma série de tarefas que são executadas em diferentes processadores.*

19

A Arte da Depuração (Abordagens)

- **FORÇA BRUTA**
  - A categoria de depuração por *força bruta* provavelmente é o método mais comum e menos eficiente de isolar a causa de um erro de software.
  - Usa-se uma filosofia do tipo "deixe que o computador descubra o erro". Por exemplo, são feitas listagens de memória (*memory dumps*), são inseridas *Flags* (por meio de instruções *WRITE*), etc.
  - Espera-se encontrar, em algum lugar do emaranhado de informações, uma pista que possa levar à causa de um erro.
  - Além disso, a massa de informações produzida possa levar ao sucesso, mais frequentemente ela conduz a tempo/esforço desperdiçados.
- **BACKTRACKING**
  - Abordagem que pode ser usada com sucesso em pequenos programas.
  - Iniciando-se no local em que o sintoma foi descoberto, o código fonte é rastreado para trás (manual) até que o local da causa seja encontrado.
  - Infelizmente, à medida que o número de linhas de código aumenta, o número de potenciais caminhos de *Backtracking* pode tornar-se incontrolavelmente alto.

20