Engenharia de Software

Tema da Aula

Teste de Software

I – Conceitos e Estratégias

Prof. Cristiano R R Portella
portella@widesoft.com.br

Conceitos
Teste e Garantia de Qualidade

Importância do Teste, segundo Deutsch:

> *"O desenvolvimento de software envolve uma série de atividades de produção, com **alta probabilidade de inserção de erros devido a falhas humanas.***
>
> *Por causa da falta de habilidade do ser humano de cumprir tarefas e de comunicar-se com perfeição, torna-se necessário garantir a qualidade de software".*

A maioria dos erros são humanos e tem origem na comunicação, entendimento e transformação das informações.

PUC CAMPINAS
Engenharia de Software

Conceitos
Teste e Garantia de Qualidade

A atividade de teste é o processo de executar um programa com a intenção de descobrir um erro.

Um bom "Caso de Teste" é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto.

Teste **não serve** para mostrar a ausência de defeitos, mas sim que **eles estão presentes**.

Durante o teste observamos as falhas.

Na Depuração (debugging) encontramos os defeitos (causa) para corrigi-los.

PUC CAMPINAS
Engenharia de Software

Terminologia

- ✓ Defeito (Fault)

  Instrução ou definição incorreta.

- ✓ Falha (Failure)

  Resultados incorretos

- ✓ Erro (Mistake)

  Falha resultante de ação humana

Fonte: IEEE Std 729, *Standard Glossary of Software Engineering Terminology*

Durante o teste observamos as falhas. Na depuração do código encontramos os defeitos (causas) para corrigi-los.

PUC CAMPINAS
Engenharia de Software

Conceitos
Teste e Garantia de Qualidade

Não existe software livre de defeitos, o que não pode servir de desculpa para não se aplicar Técnicas de Garantia de Qualidade em Software e Testes para localização/eliminação de erros.

Um valor típico é de 10 erros/KLOC.

O custo de localização e remoção de defeitos aumenta à medida em que o ciclo de desenvolvimento evolui. Quanto antes uma falha for revelada, menor o custo de reparação e maior a probabilidade de corrigi-la corretamente.

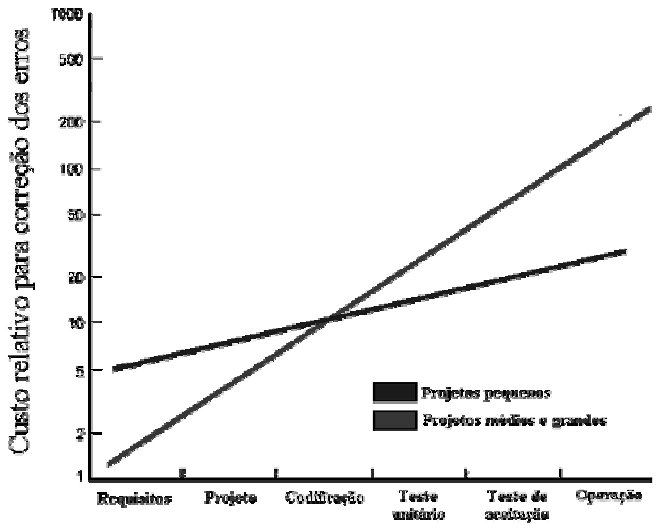
PUC CAMPINAS
Engenharia de Software

Conceitos
A Importância do Teste de Software

- ✓ Os erros são cometidos:
  - 60% nas fases iniciais do desenvolvimento
  - 40% durante a implementação
- ✓ A Maioria do erros encontra-se nas partes pouco executadas do código (esconde-se nos cantos);
- ✓ Um bom teste é, no mínimo, tão difícil quanto o desenvolvimento de software (quanto mais complexo o software, mais difícil a montagem do teste).

PUC CAMPINAS
Engenharia de Software

Custo de Correção de Erros



PUC CAMPINAS
Engenharia de Software

Terminologia

A atividade de Teste também é conhecida como Verificação e Validação (**V&V**).

A Verificação refere-se ao conjunto de atividades que garante que o software implemente corretamente uma função específica.

A Validação refere-se a um conjunto de atividades que garante que o software construído é rastreável às exigências do cliente.

PUC CAMPINAS
Engenharia de Software

Terminologia

Verificação:

Estamos construindo certo o produto ?

Validação:

Estamos construindo o produto certo ?

PUC CAMPINAS
Engenharia de Software

Atividades de Teste

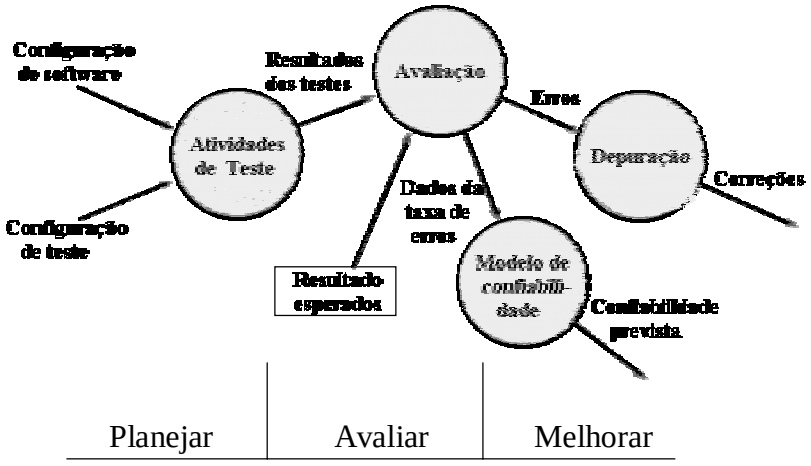
1. Planejamento
   - Definição de Padrões
   - Critérios de Adequação (Parada)
   - Modelos de Estimativa
2. *Projeto de Casos de Teste*
3. *Execução dos Casos de Teste*
4. *Análise dos Resultados Obtidos*

Técnicas

5. Documentação e Registro

PUC CAMPINAS
Engenharia de Software

Visão Detalhada do Teste (Fluxo de Atividades)



PUC CAMPINAS
Engenharia de Software

Princípios para um bom Teste

- ✓ Planejar tipo de teste
- ✓ Planejar detalhes da atividade
- ✓ Definir o procedimento de testes
- ✓ Definir os resultados esperados

} Plano de Teste

- ✓ Avaliar resultados obtidos (Obtido x Esperado)
- ✓ Melhoramento Contínuo do Processo, redefinindo técnicas e a confiabilidade prevista, através de melhoria em: Normas, Políticas, Procedimentos e Ferramentas de testagem.

PUC CAMPINAS
Engenharia de Software

Conteúdo de um Plano de Teste

- ✓ Processo de Teste
  - Descrição de cada fase do Teste (Estratégia)
- ✓ Rastreabilidade de Requisitos
  - Planejamento de teste para cada requisito
- ✓ Itens que serão Testados
  - Descrição detalhadas de cada Item que será "testado" (Modelo, Manual, Programa, etc..)
- ✓ Cronograma
  - Além do Tempo, Matriz de Alocação de Recursos x Atividades-Fases

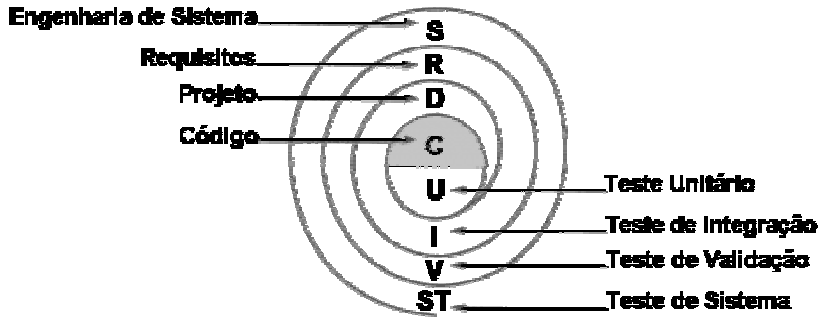
PUC CAMPINAS
Engenharia de Software

Conteúdo de um Plano de Teste

- ✓ Procedimentos de Registro
  - Definição das Métricas e Padronização dos mecanismos de registro de resultados, para que o processo de teste possa ser medido
- ✓ Requisitos de Hardware, Software e Rede
  - Lista de recursos necessários para o teste
- ✓ Descrição das Restrições
  - Restrições que afetarão o processo de teste (Ex: Deficiência de Pessoal, Treinamento de Pessoal, Aquisição de Software, etc...)

PUC CAMPINAS
Engenharia de Software

Tipos de Teste nas respectivas Fases do Desenvolvimento



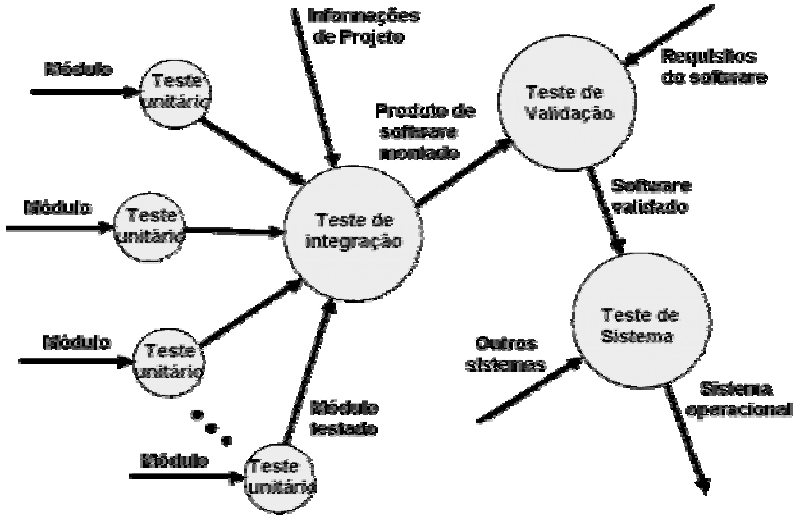
PUC CAMPINAS
Engenharia de Software

Tipos de Testes

- ✓ **Teste Unitário:** Teste dos Módulos (ou Classes) individualmente (cada unidade).
- ✓ **Teste de Integração:** Teste da Integração entre os módulos (ou classes). Teste do Projeto do Software.
- ✓ **Teste de Validação (ou aceitação):** Teste pra verificar se o produto de software atende os requisitos (conformidade com os Requisitos).
- ✓ **Teste de Sistema:** Combinação de diferentes testes para por a prova todos os diferentes elementos do sistema (foram adequadamente integrados ? Realizam corretamente as funções ?)

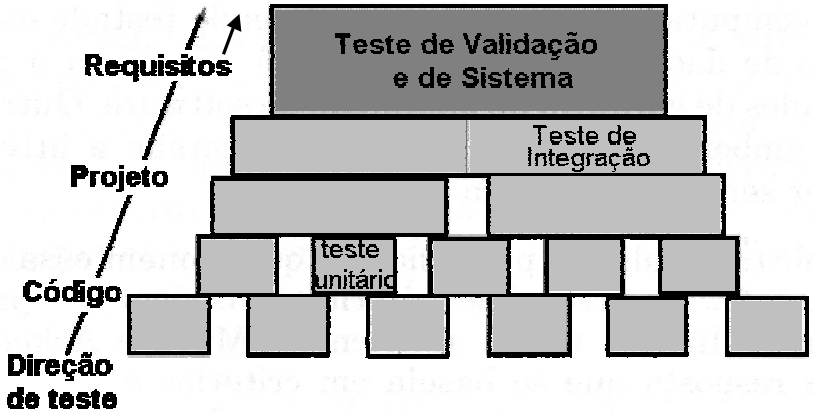
PUC CAMPINAS
Engenharia de Software

Tipos de Teste
Durante o Desenvolvimento



PUC CAMPINAS
Engenharia de Software

Progresso dos Testes



PUC CAMPINAS
Engenharia de Software

Teste Unitário

Foco: Atividade de **verificação** na menor unidade do software (módulo, classe, programa, etc..)

Abordagem Prática:

1. Aplicar Técnicas Funcionais (visão externa do produto de software – entradas e saídas)
2. Depois, complementar com técnicas estruturais (visão interna do produto de software - algoritmo)

PUC CAMPINAS
Engenharia de Software

Teste de Integração

Foco: Atividade Sistemática para **verificar** a Construção da Estrutura do software e também para a interface (comunicação) entre os módulos

✓ Porque Teste de Integração é necessário?

- Dados podem se perder na Interface entre os Módulos
- Um módulo pode ter efeito inadequado sobre outro
- Combinação de Sub-funções podem não gerar a função principal desejada
- Estruturas Globais podem afetar o software

PUC CAMPINAS
Engenharia de Software

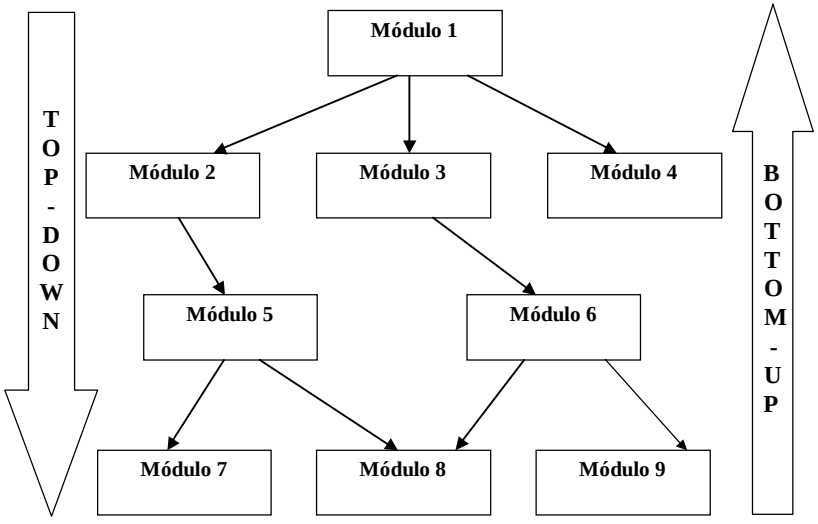
Teste de Integração

Abordagem incremental

- ✓ Teste através de segmentos de módulos que se integram;
- ✓ Complexidade controlável: módulos são integrados dois a dois;
- ✓ Três formas:
  - top-down
  - bottom-up
  - sanduíche

PUC CAMPINAS
Engenharia de Software

Integração
Top-Down x Bottom-UP



PUC CAMPINAS
Engenharia de Software

Estratégias de Teste
Abordagem Top-Down

Abordagem Top-Down:

Inicia-se a integração pelo primeiro módulo até o último da hierarquia (de cima para baixo).

✓ Duas abordagens:

- Em Largura: Integra-se, a princípio, todos os módulos subordinados
- Em Profundidade: Integra-se todos os módulos de um caminho de controle do software (que implementa uma certa funcionalidade) da estrutura do software

✓ Problema Logístico: Uso obrigatório de "**stubs**"

PUC CAMPINAS
Engenharia de Software

Estratégias de Teste
Abordagem Top-Down

✓ **Stubs**: Módulos "simplificados" que substituem outros de nível mais avançados ainda não integrados (top-down).

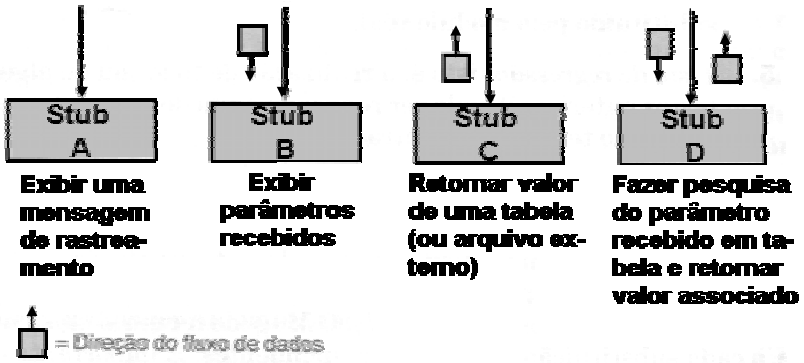
✓ Como lidar com esse problema logístico?

- Adiar a execução de alguns casos de teste que certamente causarão a chamada do módulo que ainda não foi construído;
- Criar **stubs** que simulem as principais funções do módulo não construído.

PUC CAMPINAS
Engenharia de Software

Estratégias de Teste
Abordagem Top-Down

Tipos de Stubs:



PUC CAMPINAS
Engenharia de Software

Estratégias de Teste
Abordagem Top-Down

Tipos de stubs:

1. Mostra mensagem de trace (“entrei no stub”)
2. Mostra a lista de parâmetros que foi passada (recebi a=8, b=9, x=“a:\dados.mdb”)
3. Retorna um valor, previamente armazenado em um tabela (no stub) ou em um arquivo externo
4. Recebe parâmetros, faz um busca na tabela (interna ou arquivo externo e retorna valor para o módulo chamador)

PUC CAMPINAS
Engenharia de Software

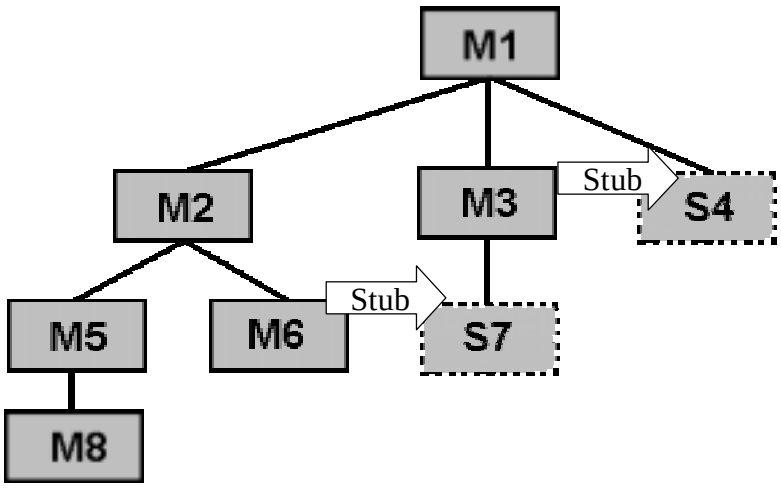
Estratégias de Teste
Abordagem Top-Down

Processo de Integração (incremental):

1. Testa-se o primeiro módulo
2. A cada Passo:
   - Substitui-se um "stub" por um novo módulo subordinado
   - Módulo testado permanece

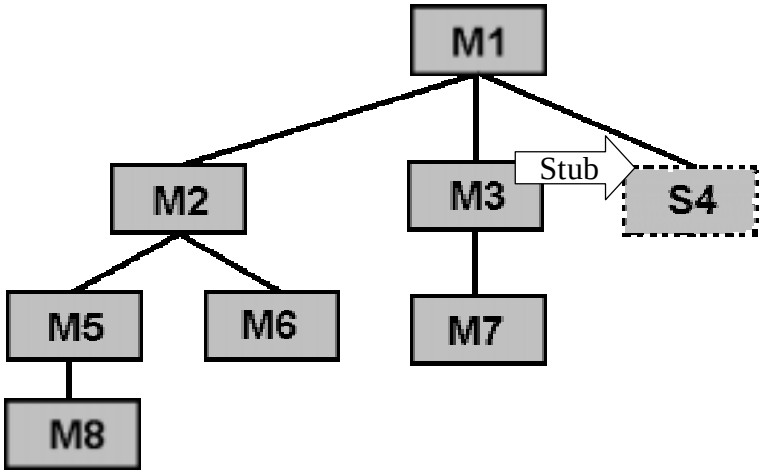
PUC CAMPINAS
Engenharia de Software

Integração Top-Down
Profundidade 1/3



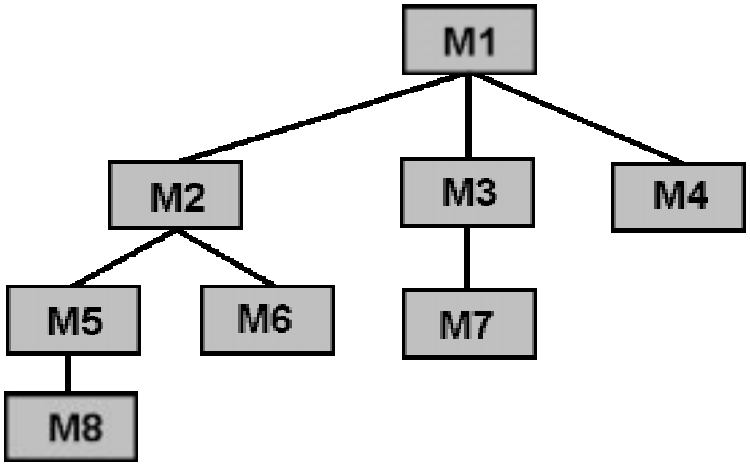
PUC CAMPINAS
Engenharia de Software

Integração Top-Down
Profundidade 2/3



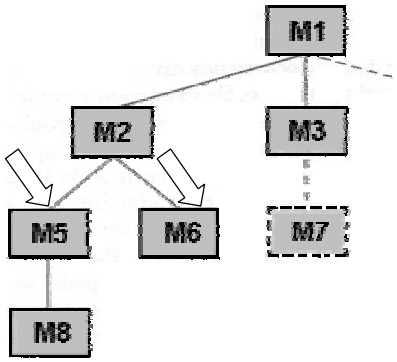
PUC CAMPINAS
Engenharia de Software

Integração Top-Down
Profundidade 3/3



PUC CAMPINAS
Engenharia de Software

Integração Top-Down
Definição da Seqüência de Teste



Seqüência de teste:
M1 – M2
M1 – M2 – M5
M1 – M2 – M5 – M8
M1 – M2 – M6;

Mas se **M6** for necessário para que **M2** funcione corretamente:
M1 – M2
M1 – M2 – M6
M1 – M2 – M5
M1 – M2 – M5 – M8 .

PUC CAMPINAS
Engenharia de Software

Estratégias de Teste
Abordagem Bottom-Up

Abordagem Bottom-Up:

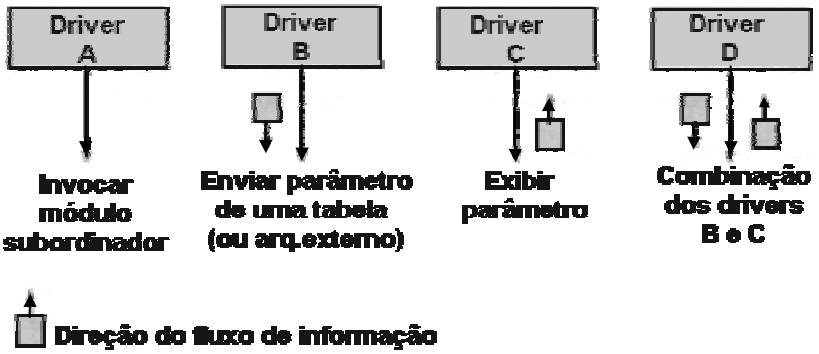
Módulos são integrados partindo-se do último da hierarquia (de baixo para cima).

- ✓ Novo problema logístico: Um "**driver**" deve ser providenciado para coordenar as entradas, saídas e chamadas do módulo (substituir stubs por driver).
- ✓ **Driver**: Programa de controle escrito para coordenar a entrada e saída do Caso de Teste (navegação).

PUC CAMPINAS
Engenharia de Software

Estratégias de Teste
Abordagem Bottom-Up

Tipos de Drivers:



PUC CAMPINAS
Engenharia de Software

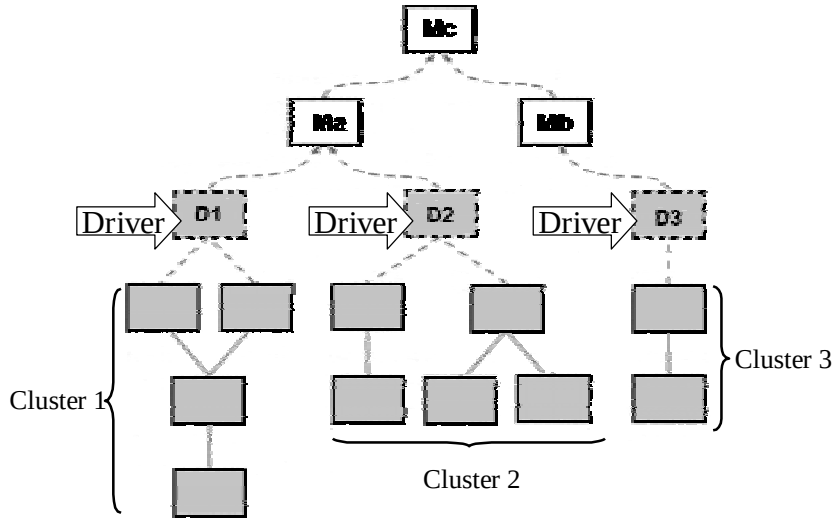
Estratégias de Teste
Abordagem Bottom-Up

Processo de Integração:

1. Módulo de nível mais baixo são mapeados em "clusters" (conjunto de módulos que executam alguma função do software)
2. Driver coordena a entrada e saída dos dados
3. Cluster é testado (mesmo que incompleto)
4. Troca-se o driver pelo módulo hierarquicamente superior (integra-se cada "cluster" pouco a pouco)

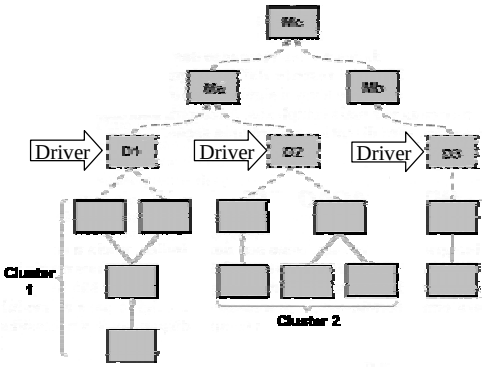
PUC CAMPINAS
Engenharia de Software

Estratégias de Teste
Abordagem Bottom-Up



PUC CAMPINAS
Engenharia de Software

Estratégias de Teste
Abordagem Bottom-Up



1- Driver D1 é usado para testar *Cluster 1*.
2- Driver D2 é usado para testar *Cluster 2*.
3- Quando o bloco Ma estiver pronto, ele substituirá os drivers D1 e D2.
4- Driver D3 é usado para testar *Cluster 3*.
5- Quando o bloco Mb estiver pronto ele substituirá o Driver D3.
6- O Driver D4 será criado para testar Ma e Mb.
7- Quando o bloco Mc estiver pronto ele substituirá o Driver D4 integrando Ma e Mb

PUC CAMPINAS
Engenharia de Software

Estratégias de Teste
Top-Down ou Botton-Up

	Top-Down	**Botton-Up**
Desvan-tagens	Necessidade de criar stubs	O programa não existe como entidade até que o último módulo seja adicionado. Necessidade de criar drivers (mais fáceis que stubs)
Vantagens	Testa antes as principais funções de controle.	Projeto de Caso de Teste mais fácil pela ausência de stubs.

PUC CAMPINAS
Engenharia de Software

Estratégias de Teste
Top-Down ou Botton-Up

Definir os módulos críticos e dar prioridades a eles (quanto mais rápidos testa-los, melhor).

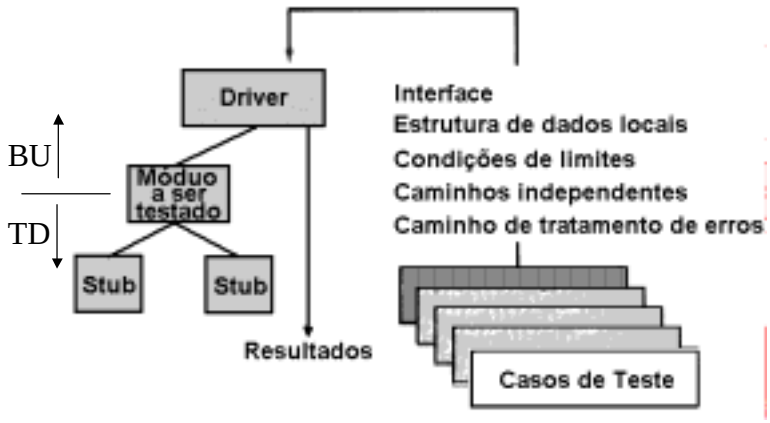
Dependendo de sua posição na estrutura do produto, escolher a abordagem.

Módulos críticos:

- ✓ Abordam diversos requisitos do software;
- ✓ Tem elevado nível de controle (ponto alto na estrutura);
- ✓ É complexo ou propenso a erros; e
- ✓ Tem restrições de desempenho definidas.

PUC CAMPINAS
Engenharia de Software

Estratégias de Teste
Teste de Sistema



PUC CAMPINAS
Engenharia de Software

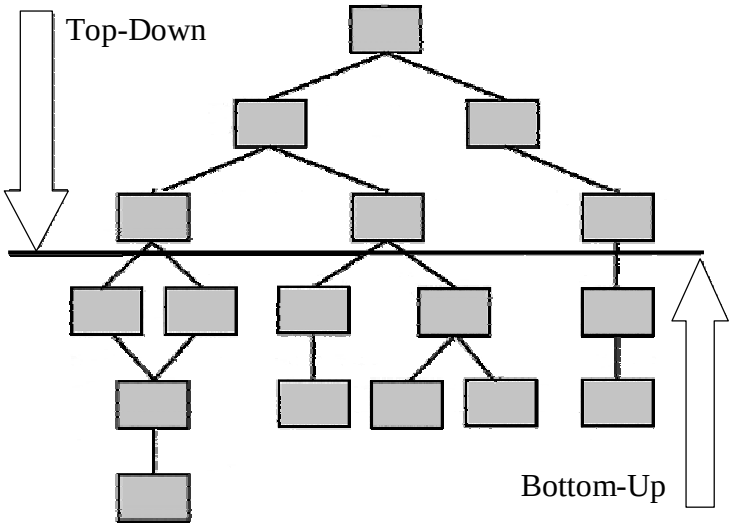
Estratégias de Teste
Abordagem Alternativa (Sanduíche)

Abordagem Combinada ou Sanduíche

- ✓ Mistura as melhores características das anteriores
- ✓ Deve-se avaliar sua aplicabilidade caso a caso
- ✓ Define-se um linha base (ponto de inflexão) na estrutura de integração dos módulos:
  - Acima da linha: TOP-DOWN
  - Abaixo da linha: BOTTOM-UP

PUC CAMPINAS
Engenharia de Software

Estratégias de Teste
Abordagem Sanduíche



PUC CAMPINAS
Engenharia de Software

Projeto de Casos de Teste

Caso de Teste: Entrada, Saída Esperada.

- ✓ Tão difícil quanto o projeto do produto
- ✓ Poucos gostam de teste; menos pessoas gostam de projetar Casos de Teste
  - Software é lógico; Teste é ainda mais abstrato;
  - Esforço de teste parece desperdiçado se não forem expostas falhas no software;

PUC CAMPINAS
Engenharia de Software

Projeto de Casos de Teste

Teste Exaustivo é o Ideal: Todos os erros serão identificados e corrigidos (porém é impraticável).

1ª Lei: Paradoxo do Pesticida

Todo método usado para prevenir erros/defeitos é ineficaz para algum tipo de erro/defeito.

2ª Lei: Barreira da Complexidade

A complexidade do software (e consequentemente dos erros) cresce em função dos limites de nossa habilidade em gerenciar aquela complexidade.

PUC CAMPINAS
Engenharia de Software

Executar os Casos de Teste

- ✓ Preparar os Scripts de Teste.
- ✓ Executar o Conjunto de Casos de teste (Test Suite) "em batch" .
- ✓ Armazenar o "Test Suite".
- ✓ Ferramentas automatizadas de teste aumentam a produtividade da execução dos casos de teste.

PUC CAMPINAS
Engenharia de Software

Análise dos Resultados

- ✓ Verificar cada resultado obtido contra o esperado;
- ✓ Anotar todas as ocorrências (não conformidades);
- ✓ Resolver cada ocorrência individualmente, considerando as possibilidades:
  - Erro de Codificação
  - Erro de Análise e/ou Especificação
  - Erros de Teste.

PUC CAMPINAS
Engenharia de Software

Depuração

Quando um teste bem sucedido revela uma falha, a depuração (debugging) é o processo de localização do defeito e sua remoção.

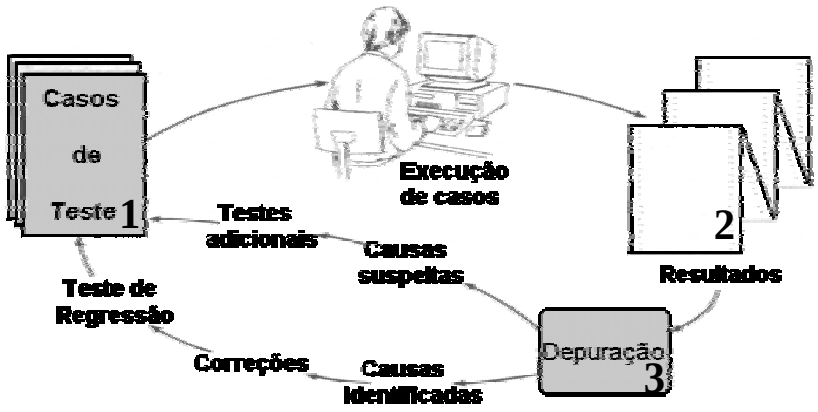
Pode ser um processo empírico, pois muitas vezes a manifestação externa do erro (falha) e sua causa interna (defeito) não tem relação óbvia entre si.

O processo de depuração tenta ligar o sintoma a uma causa provável, que se encontrada será corrigida.

Se a causa não for descoberta, será projetado novo Caso de Teste para validar uma suspeita de causa da falha.

PUC CAMPINAS
Engenharia de Software

Depuração



PUC CAMPINAS
Engenharia de Software

Teste de Regressão

Teste de Regressão:

- Repetição dos testes já executados, a fim de garantir que as novas modificações não introduziram novos defeitos em aspectos do software que já haviam sido testados e depurados.
- Ferramentas de testagem permitem que os testes de regressão sejam realizados de maneira automática e rápida.

PUC CAMPINAS
Engenharia de Software

Depuração

O processo de depuração torna-se particularmente difícil quando:

- ✓ Sintoma e causa estão distantes;
- ✓ O sintoma desaparece (temporariamente) quando outro erro é corrigido;
- ✓ O sintoma é causado por “não-erro” (por exemplo o resultado de um arredondamento em cascata);
- ✓ Sintoma causado por erro humano (difícil de rastrear);
- ✓ Sintoma causado por erro de “timing” (executado no momento errado);

PUC CAMPINAS
Engenharia de Software

Depuração

O processo de depuração torna-se particularmente difícil quando:

- ✓ Condições de entrada difíceis de reproduzir com precisão (por exemplo em aplicações de tempo real);
- ✓ Sintoma intermitente (particularmente comum em sistemas embutidos); e
- ✓ Sintoma tem causas distribuídas por diferentes tarefas (múltiplas causas concorrentes).

PUC CAMPINAS
Engenharia de Software

Depuração

Geralmente, à medida em que passa o tempo de depuração, os erros remanescentes são mais sutis, demandando mais esforço ou diminuindo a probabilidade de sua localização.

PUC CAMPINAS
Engenharia de Software

Depuração

Abordagens de depuração:

1. Força Bruta:
   Método mais comum e menos eficiente, deixa que o próprio computador descubra o erro, usando traces e instruções inseridas para ajudar a determinar o momento da falha.
2. Backtracking:
   Abordagem usada em pequenos programas. A pesquisa inicia-se no local onde a falha foi descoberta; rastreia-se o código para trás. A complexidade do código pode aumentar muito o número de caminhos a serem rastreados.

PUC CAMPINAS
Engenharia de Software

Depuração

Abordagens de depuração:

3. Eliminação da causa:

Uma hipótese de causa é imaginada e um Caso de Teste é montado para provar ou refutar a hipótese. Uma lista de todas as possíveis causas é gerada.

PUC CAMPINAS
Engenharia de Software

Depuração

3. Eliminação da causa:

O que é _____
não é _____
Quando é _____
não é _____
Onde é _____
não é _____
Em que extensão é _____
não é _____

The Method (Brown & Sampson).

PUC CAMPINAS
Engenharia de Software

Depuração

A correção de um defeito pode introduzir outras falhas. Três perguntas simples (Van Vleck) devem ser feitas ao se remover o defeito:

1. A causa do defeito é reproduzida em outra(s) partes do software (bloco padrão copiado ou padrão de programação) ?
2. A correção do defeito pode introduzir nova falha (parte do programa fortemente acoplada a estruturas lógicas ou estruturas da informação) ?
3. O que poderia ser feito para eliminar essa falha desde o princípio (abordagem de Garantia de Qualidade de Software) ?

PUC CAMPINAS
Engenharia de Software

Revisões Técnicas Formais (FTR)

Também chamadas de walkthroughs, inspeções, revisões round-robin etc é uma técnica de garantia da qualidade de software (atividade guarda-chuva), que tem os seguintes objetivos:

- ✓ Descobrir erros de função, lógica ou implementação
- ✓ Verificar se o software atende aos requisitos
- ✓ Verificar se documentação técnica atende padrões e formalismo
- ✓ Obter software desenvolvido de maneira estruturada e uniforme
- ✓ Tornar os projetos mais “administráveis”.

PUC CAMPINAS
Engenharia de Software

Revisões Técnicas Formais (FTR)

Reunião de Revisão Técnica Formal:

- ✓ Duração máxima de 2 horas;
- ✓ De 3 a 5 participantes;
- ✓ Somente desenvolvedores (sem chefias);
- ✓ Analisar o produto e não o desenvolvedor;
- ✓ Definir “líder” e “anotador”;
- ✓ Preparar material para os participantes; e
- ✓ Apontar os problemas e não tentar resolve-los.

PUC CAMPINAS
Engenharia de Software

Exercício

Em grupo de 4 alunos, crie um **formulário** que será usado para “Plano de Teste”, contendo **no mínimo**, as seguintes informações:

PUC CAMPINAS
Engenharia de Software

Exercício

- ✓ Nome do Sistema;
- ✓ Nome do(s) módulos em teste (ou produto todo);
- ✓ Fase do ciclo de vida em que cada teste será realizado;
- ✓ Técnicas empregadas e respectivas ferramentas;
- ✓ Responsável(eis) pela aplicação do teste;
- ✓ Cronograma de teste (início-fim-duração);
- ✓ Responsável(eis) pelo registro dos resultados;
- ✓ Responsável(eis) pela verificação e aprovação;
- ✓ Critérios para a conclusão de cada fase; e
- ✓ Normas/padrões a serem seguidos,

PUC CAMPINAS
Engenharia de Software

Exercício

Seu trabalho é:

- ✓ Dispor as informações no melhor arranjo possível.
- ✓ Incluir as informações que o grupo entender necessário (com certeza elas existem).

  Use o material de aula, a bibliografia recomendada e a criatividade, para incluir campos necessários ao formulário.
- ✓ Fazer um teste (teórico) de aplicação do formulário.