Verificação e Validação (V & V)

- Objetivo: assegurar que o software que o software
  - cumpra as suas especificações e
  - atenda às necessidades dos usuários e clientes.
- Verificação:
  - “Estamos construindo certo o produto?”
  - O software deve está de acordo com a sua especificação.
- Validação:
  - “Estamos construindo o produto certo?”
  - O software deve atender às necessidades dos usuários.
- Ocorrem em todo o ciclo de vida completo
  - Revisões de requisitos, revisões de design, testes de código

O que deve ser verificado e validado?

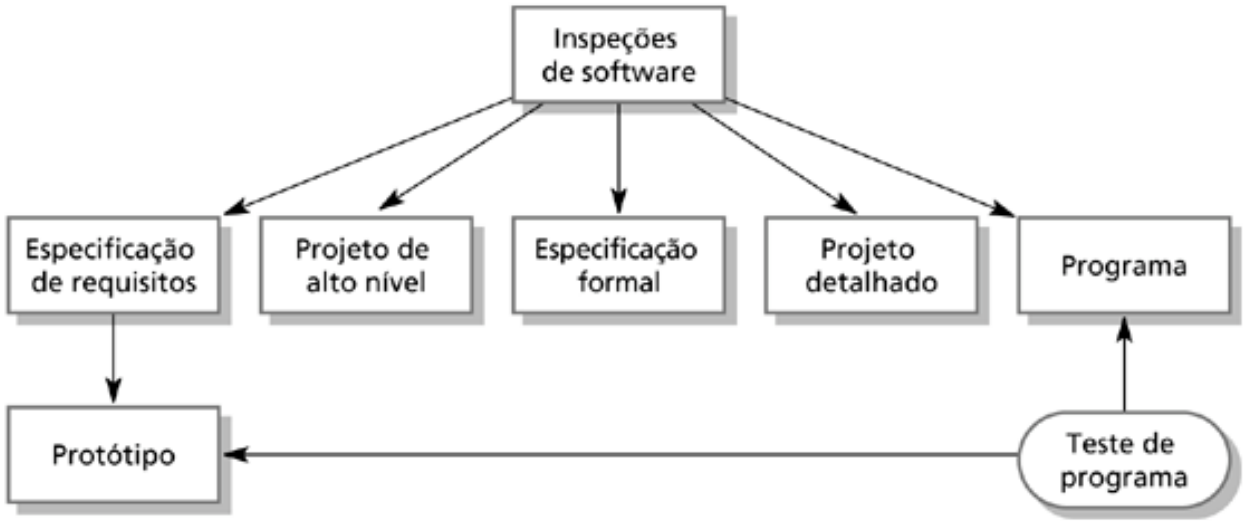
- Fatores de Qualidade Operacionais
  - Correção
  - Eficiência ou desempenho
  - Robustez
  - Confiabilidade
  - Usabilidade
  - Utilidade e validade
- Fatores de Qualidade de Revisão
  - relacionados com a manutenção, evolução e avaliação do software
- Fatores de Qualidade de Transição
  - relacionados com a instalação, reutilização e interação com outros produtos

Técnicas de V & V (Sommerville)

- Inspeções de software (V & V estática)
  - Análise da documentação e código fonte do software
  - Pode ser auxiliado por ferramentas de depuração
- Testes de software (V & V dinâmica)
  - O programa ou um protótipo devem ser executados
  - Casos de testes deve ser elaborados: dados de entrada e comportamento esperado.

Figura 22.1

Verificação e validação dinâmica e estática.

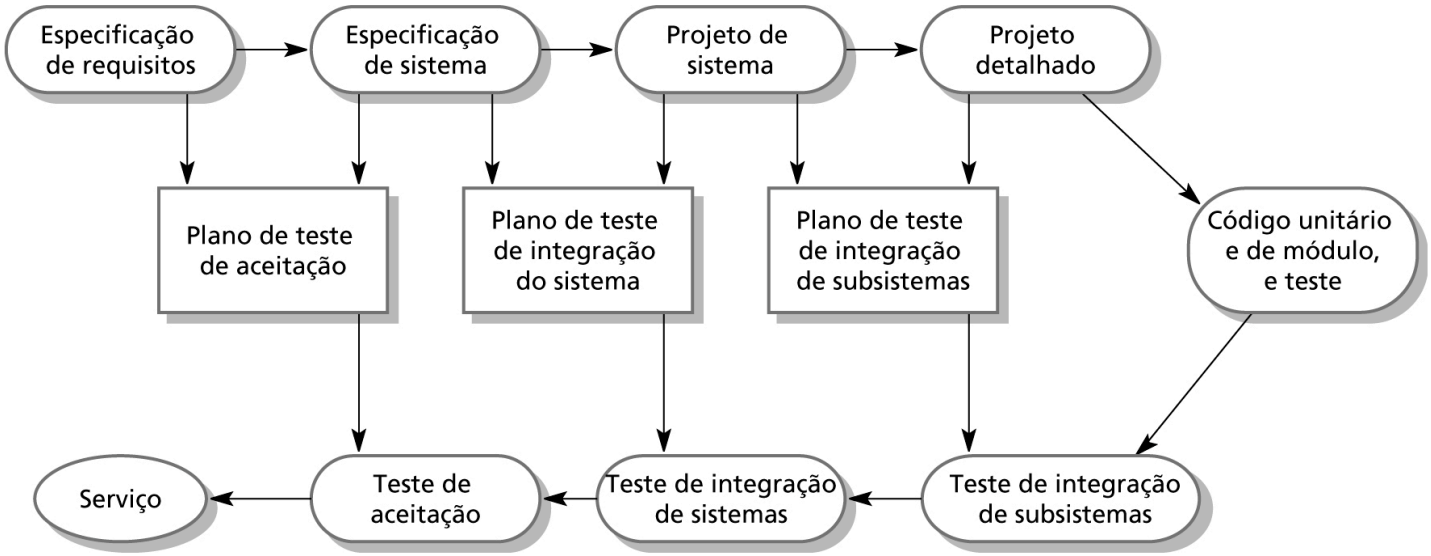


Fonte: Ian Sommerville, 2007

Plano de V&V

- O processo de V&V ocorre durante todo o ciclo de vida
- Precisa ser planejado em conjunto com outras atividades do processo de software

Figura 22.3 Plano de teste como ligação entre o desenvolvimento e os testes.



Fonte:
Ian Sommerville, 2007

Inspeções de Software

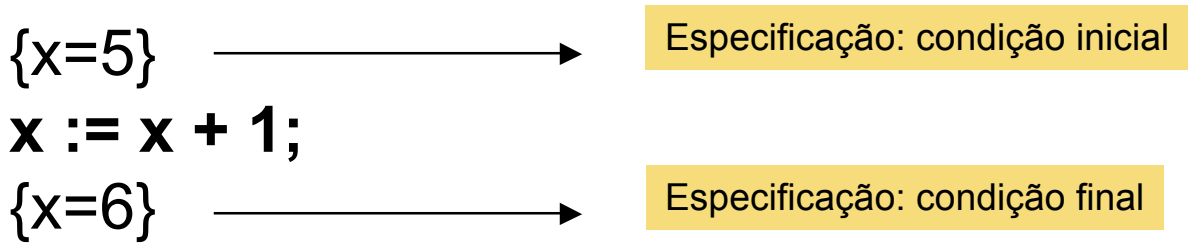
- Características
  - Técnica preventiva – permite a V & V antes do software ser codificado
  - Mais barata
  - Baseada na experiência do inspetor
  - Mais aplicada a fatores de revisão e transição
  - Pouco eficaz para fatores operacionais
- Aplicações mais comuns
  - Inspeção de programa fonte (estática e dinâmica)
  - Inspeção de documentos e modelos
  - Desenvolvimento Cleanroom

Abordagens para verificação da correção

- Inspeções analíticas
  - Estática
    - Rastreamento do código fonte (walkthrough) – percorre-se o código executando-o mentalmente.
  - Automatizada (dinâmica)
    - Depuração – execução passo-a-passo e visualização de variáveis do programa
- Testes de correção de programas
  - Testes de unidade
  - Testes de integração
- Prova Formal de Programas
  - Utiliza métodos formais de desenvolvimento de software – técnicas de especificação, transformação e prova formal

Prova Formal de Programas

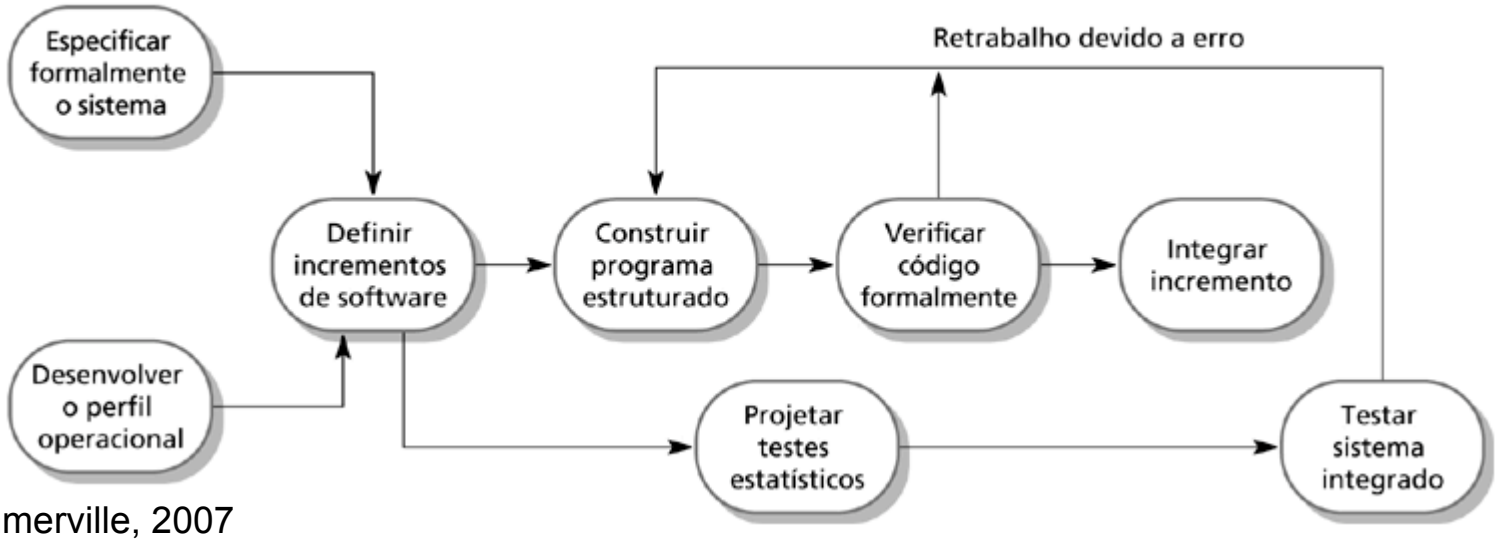
- Utilizada em métodos formais
- Linguagens de programação são definidas formalmente (sintaxe e semântica formal)
- Linguagens de especificação formais também
- Pode-se provar matematicamente que um programa está correto em relação à especificação formal



Desenvolvimento Cleanroom

- Baseado na idéia de sala limpa no desenvolvimento de circuitos integrados.
- A filosofia é prevenção ao invés de remoção de defeitos.
- Especificação formal usando um modelo de transição de estados.
- Desenvolvimento incremental onde o cliente prioriza os incrementos.
- Programação estruturada – controle limitado e construções abstratas são usadas no programa.
- Verificação estática usando inspeções rigorosas.

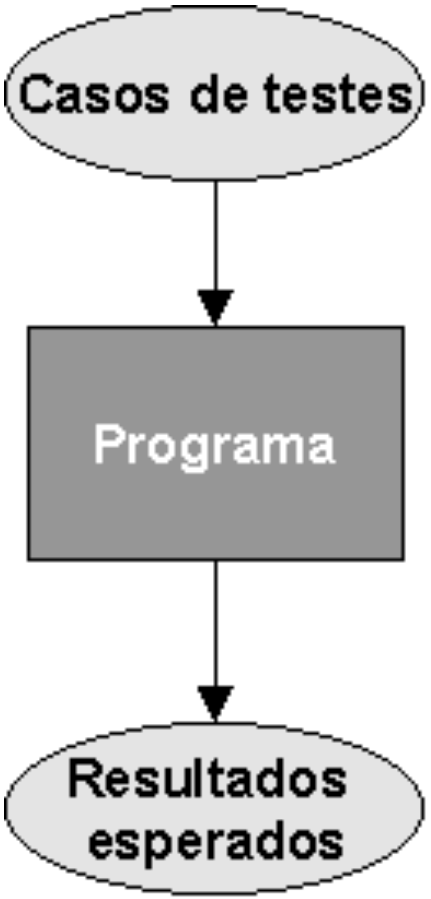
Figura 22.6 Processo de desenvolvimento Cleanroom.



Fonte:
Ian Sommerville, 2007

Testes de software

- Elaboração de casos de testes baseados na especificação funcional
  - Dados de entradas
  - Comportamento esperado
- Podem ser classificados
  - Quanto ao objetivo
  - Quanto ao escopo
  - Quanto ao método
- Aplicações em fatores operacionais
  - Correção
  - Usabilidade
  - Desempenho
  - Robustez



Tipos de testes quanto ao objetivo

- Testes de Defeitos
  - Tem por objetivo encontrar defeitos – inconsistências entre o programa e a sua especificação.
  - Verifica a correção – conhecido também por **testes de correção**
  - Normalmente realizados com protótipos funcionais
- Testes de Validação
  - Utilizado para demonstrar ao desenvolvedor e ao cliente do sistema que o software atende aos seus requisitos.
  - Um teste bem sucedido visa mostrar que o sistema opera conforme especificado pelo cliente.

Tipos de testes quanto ao método

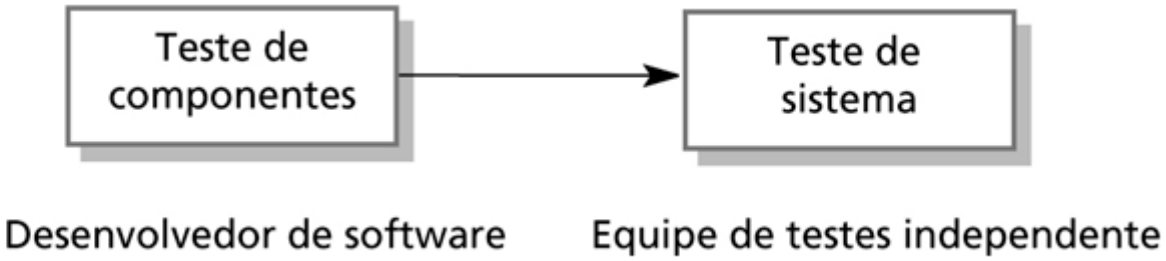
- Testes Controlados
  - Realizados em laboratório ou em condições operacionais sob a supervisão de um testador
  - Baseados na geração de **casos de testes**
  - Podem ser realizados com protótipos funcionais
- Testes Estatísticos
  - Utilizados para verificar como o software nas condições operacionais (versão beta)
  - Avalia desempenho, robustez, confiabilidade, etc
  - Utiliza programa de monitoramento e “logs” com ocorrências operacionais.
  - Exemplos de medições:
    - Número de falhas observadas
    - Tempos de resposta
    - Tempos de execução

Tipos de testes quanto ao escopo

- Testes de Unidade (componente)
  - Conhecidos como testes em ponto pequeno
  - São testadas as unidades individuais – funções, objetos, componentes.
- Testes de Integração (sistema)
  - Conhecidos como testes em ponto grande
  - Os componentes são integrados e o conjunto maior é testado – módulo e sub-sistemas.

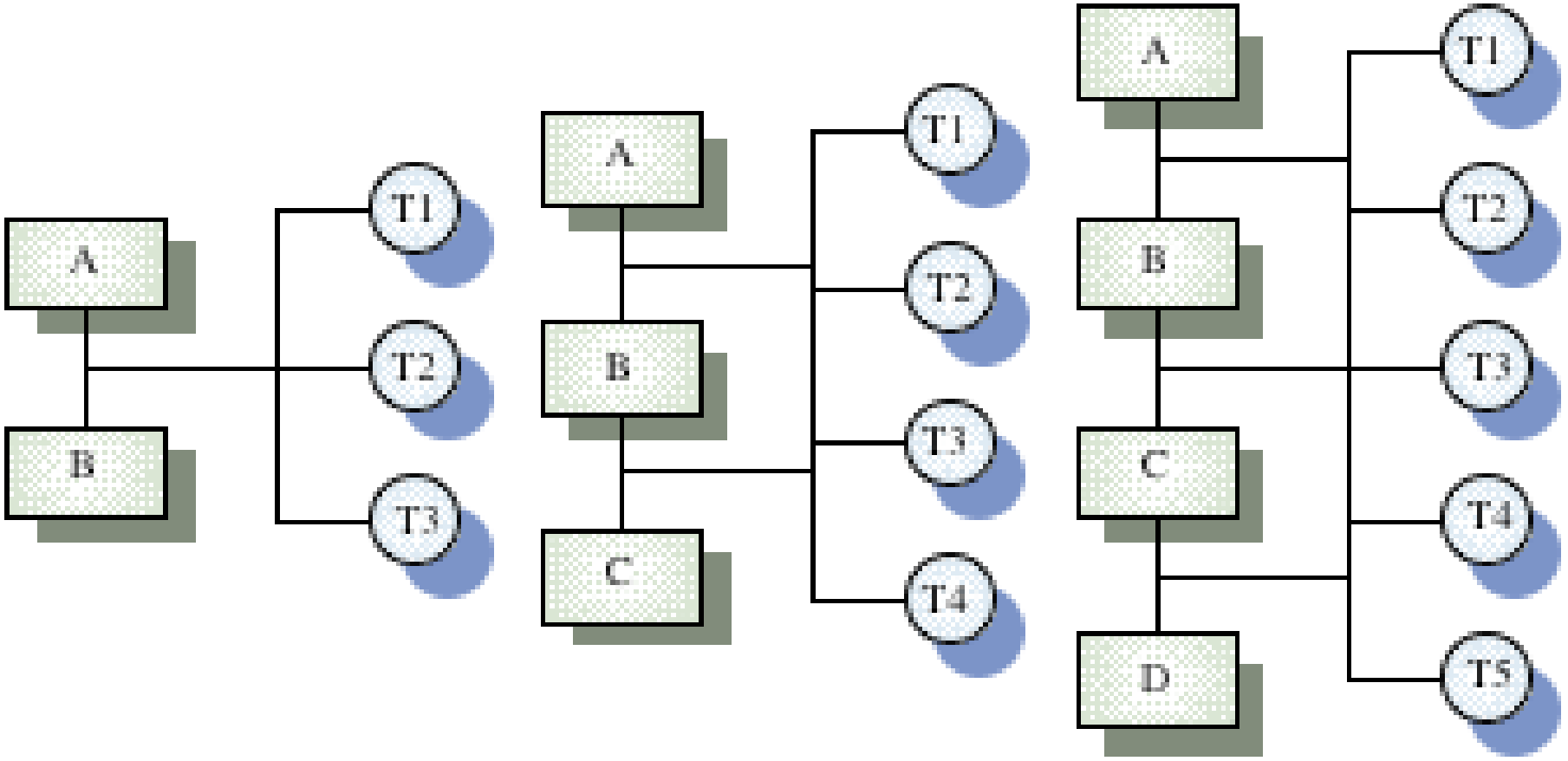
Figura 23.1

Fases de teste.



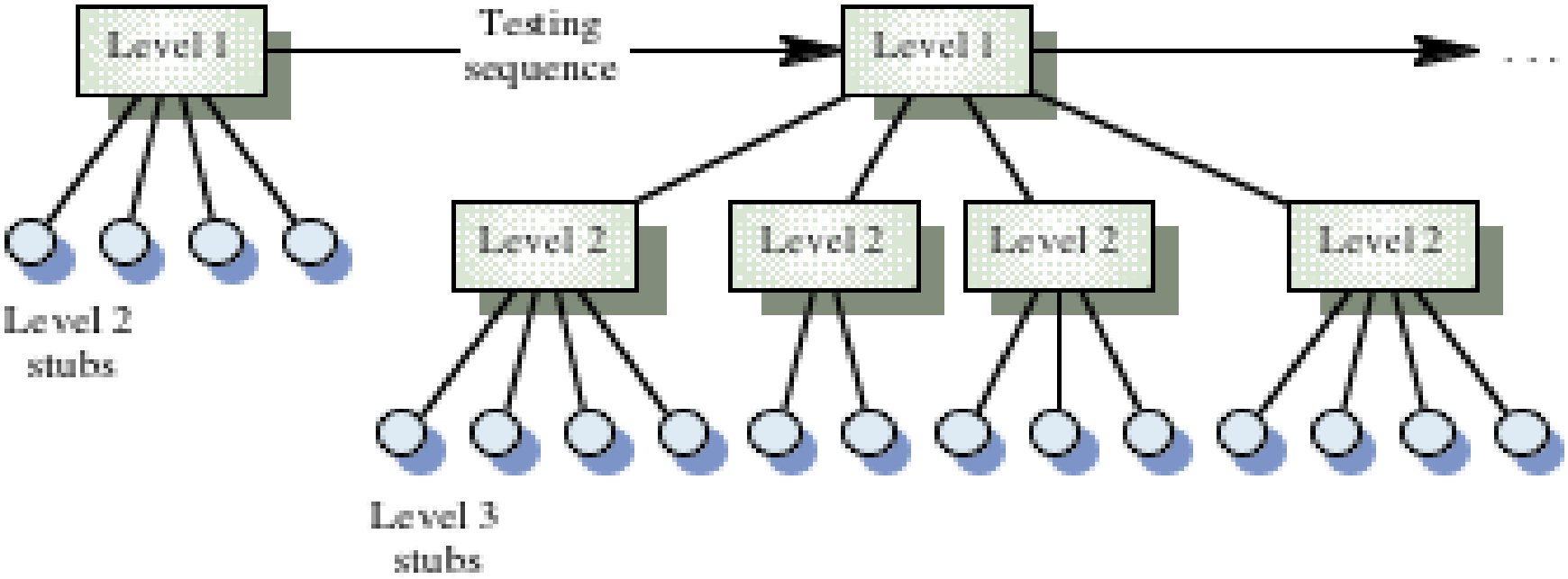
Fonte:
Ian Sommerville, 2007

Testes de integração incremental



Testes de integração Top-down

- Começa pelos componentes de alto nível e vai descendo na hieraquia de componentes, de acordo com a arquitetura do software.
- Os sub-componentes são representados por *stubs*.
- Stubs tem a mesma interface, mas não precisa ter funcionalidade. Basta retornar valores esperados.



Testes de integração Bottom-up

- Integra unidades já testadas em módulos de mais alto nível, de acordo com a arquitetura.