Engenharia de Software II / Qualidade e Teste de Software

Aula 06: V&V: Verificação e Validação, Modelo em V e Definição Formal de Testes

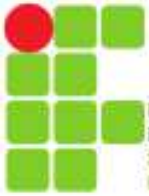
Breno Lisi Romano

http://sites.google.com/site/blromano

Instituto Federal de São Paulo – IFSP São João da Boa Vista

Bacharelado em Ciência da Computação – BCC (ENSC6)

Tecnologia em Sistemas para Internet – TSI (QTSI6)

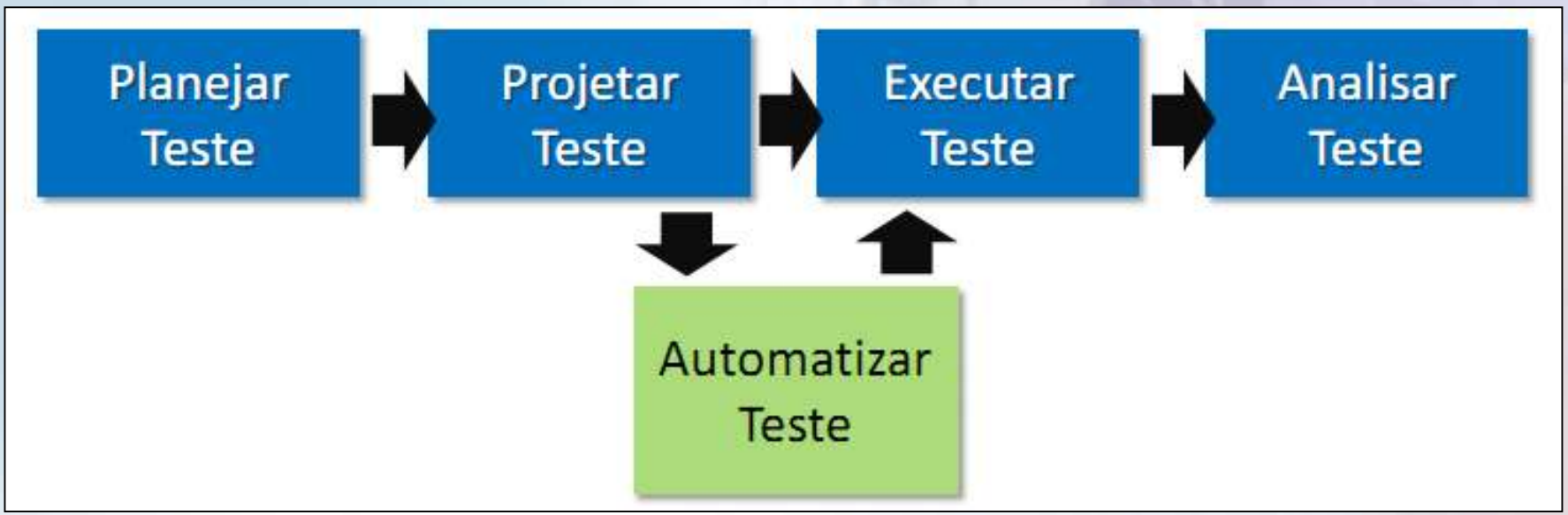


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Revisão: Teste de Software

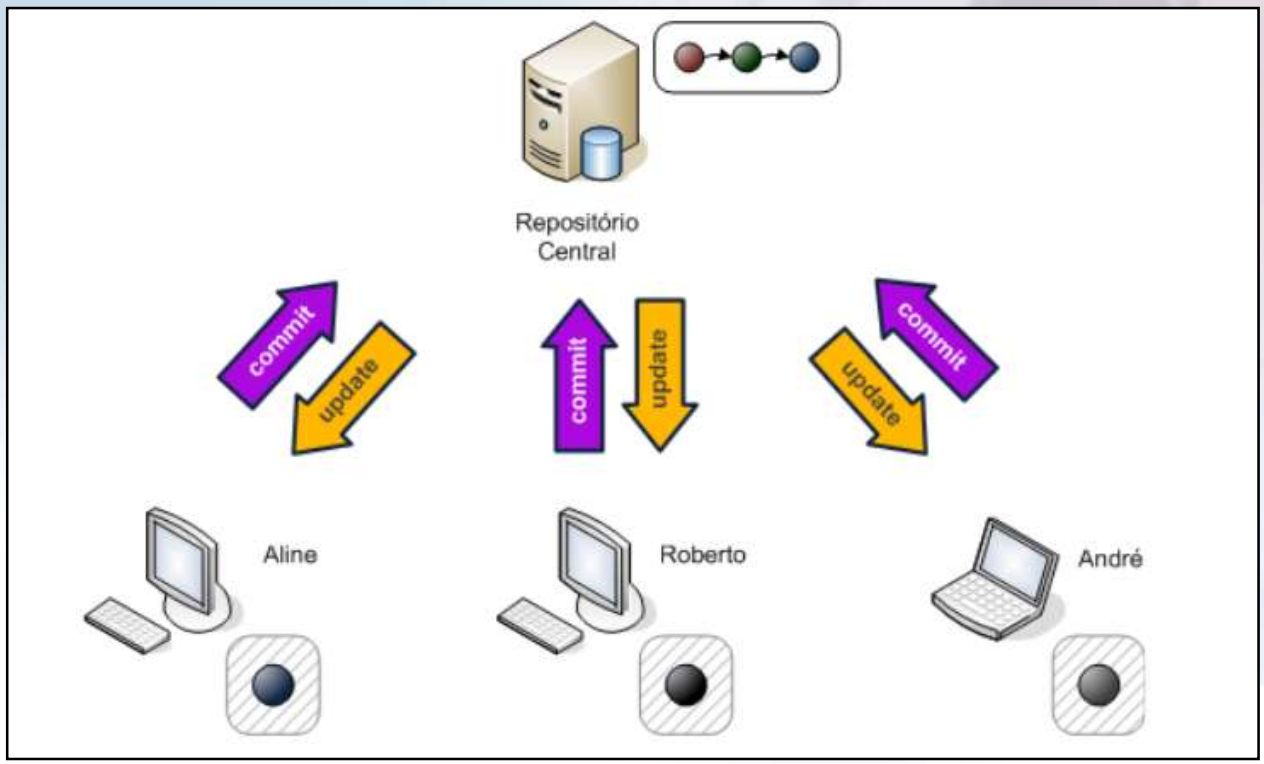
- Representam uma oportunidade de detectar **defeitos** antes do software ser entregue aos usuários
- A atividade de testes pode ser feita de forma **manual** e/ou **automática** e tem por objetivos:
  - **Detectar Erros** para **Eliminar** os **Defeitos** e **Evitar** as **Falhas**
  - **Produzir casos de teste** que tenham elevadas probabilidades de **revelar um defeito ainda não descoberto**, com uma quantidade mínima de tempo e esforço
  - **Comparar o resultado dos testes com os resultados esperados** → produzir uma indicação da qualidade e da confiabilidade do software. Quando há diferenças, inicia-se um processo de depuração para descobrir a causa

Revisão: Processo Básico de Teste de Software



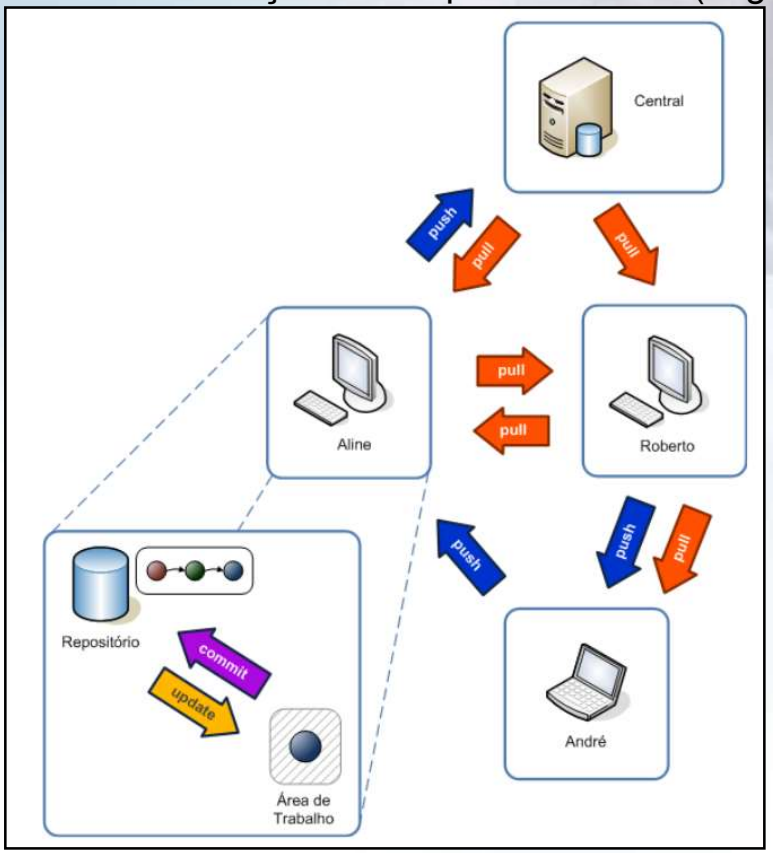
Revisão: Controle de Versões Centralizado

- **Único repositório centralizado com várias cópias de Áreas de Trabalho**
  - Comunicação entre uma área de trabalho e outra passa obrigatoriamente pelo repositório central

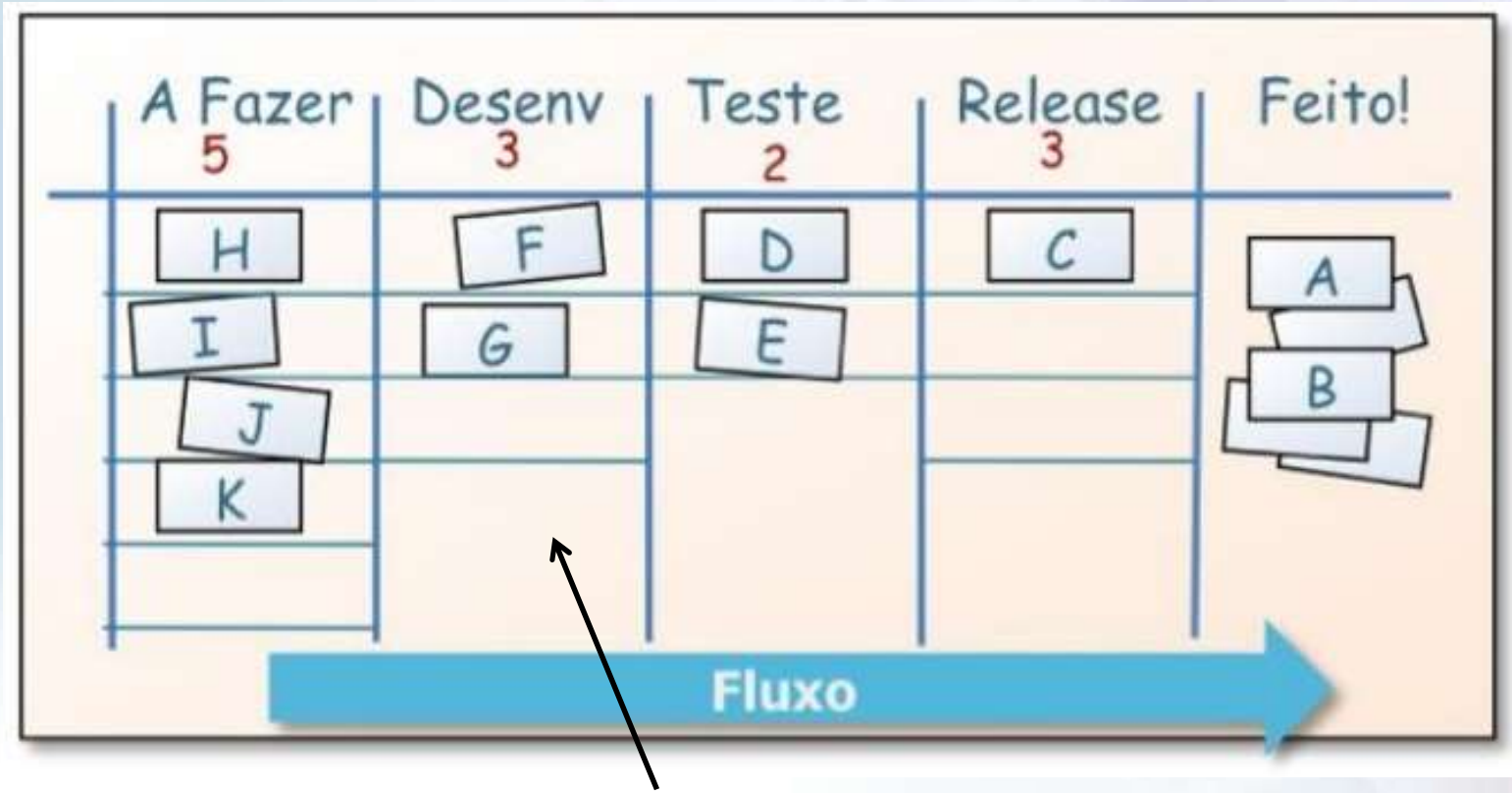


Revisão: Controle de Versões Distribuído

- Um Repositório pode se comunicar com qualquer outro através das operações básicas: **Pull** e **Push**
  - **Pull (Puxar) ou Fetch:** Atualiza o repositório local (destino) com todas as alterações feitas em outro repositório (origem)
  - **Push (Empurrar):** Envia as alterações do repositório local (origem) para um outro repositório (destino)



Revisão: Kanban



WIP (Work in Process)

https://sbv.ifsp.edu.br/kanboard/

Revisão: Testlink

- **Ferramenta para Gerenciamento de Testes:** TestLink
  - Link: https://testlink.sbv.ifsp.edu.br/login.php

Verificação e Validação

- O desenvolvimento de software está sujeito a diversos tipos de problemas, os quais acabam resultando na obtenção de um produto diferente daquele que se esperava

- Muitos fatores podem ser identificados como causas de tais problemas, mas a maioria deles tem uma única origem: **erro humano** (Delamaro et al., 2007).

- As atividades de **Verificação e Validação (V&V)** visam garantir que:
  - o software vem sendo desenvolvido corretamente
  - o software que vem sendo desenvolvido é o software correto

Conceitos Básicos

- **Verificação:**
  - Processo de avaliação de um sistema ou componente para determinar se os **artefatos** produzidos **satisfazem** às **especificações** determinadas no início da fase
  - **"Você construiu corretamente?"**

- **Validação:**
  - Processo de avaliação para determinar se o **sistema atende** as **necessidades** e **requisitos** dos usuários
  - **"Você construiu o sistema correto?"**

- **Testes:**
  - Processo de exercitar um sistema ou componente para **validar** que este **satisfaz** os **requisitos** e para **verificar** **para identificar defeitos**

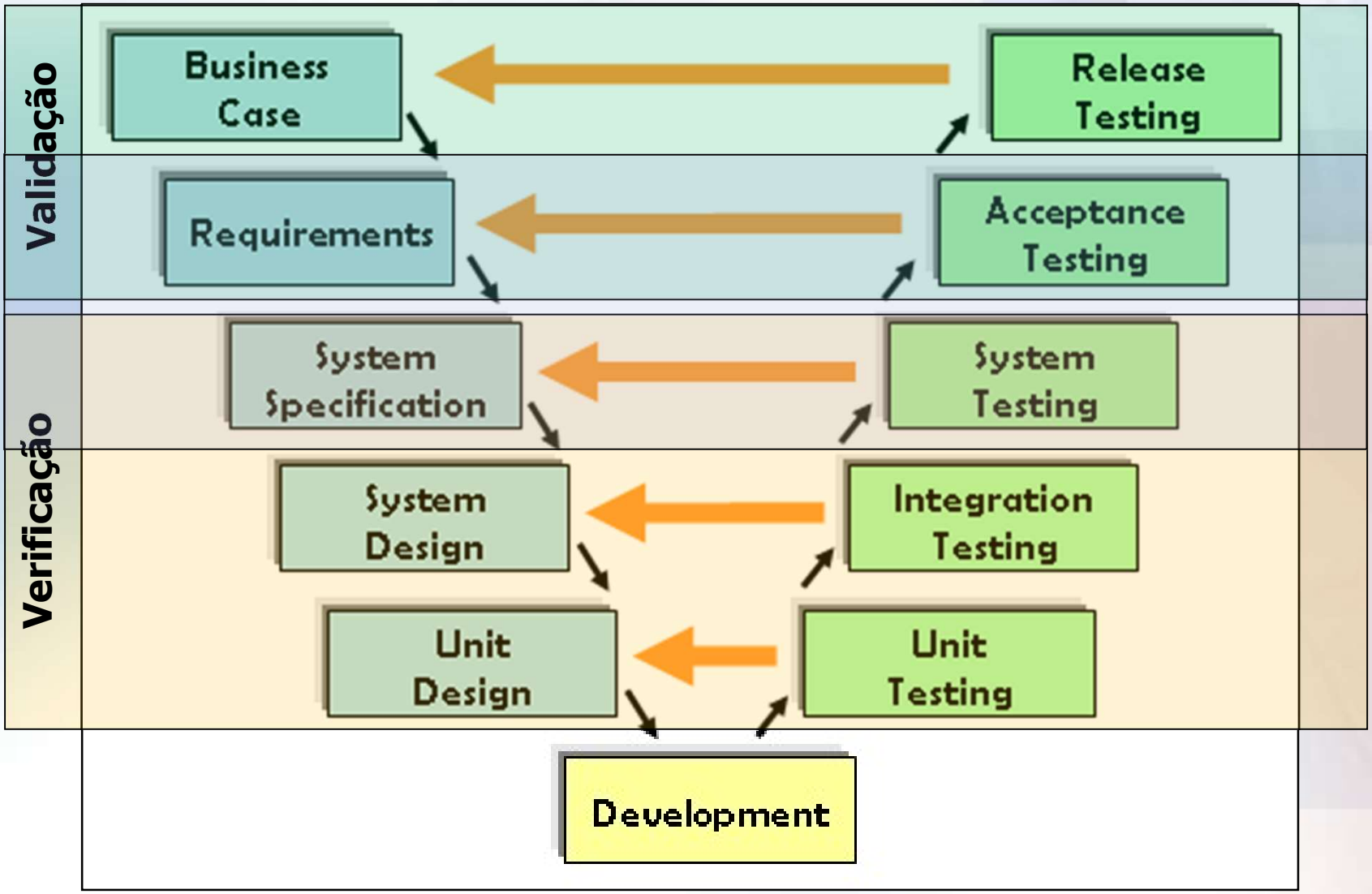
V&V: Estática x Dinâmica

- As atividades de V&V costumam ser divididas em estáticas e dinâmicas
- As estáticas não requerem a execução ou mesmo a existência de um programa ou modelo executável para serem realizadas
- As dinâmicas se baseiam na execução de um programa ou modelo (Delamaro et al., 2007)

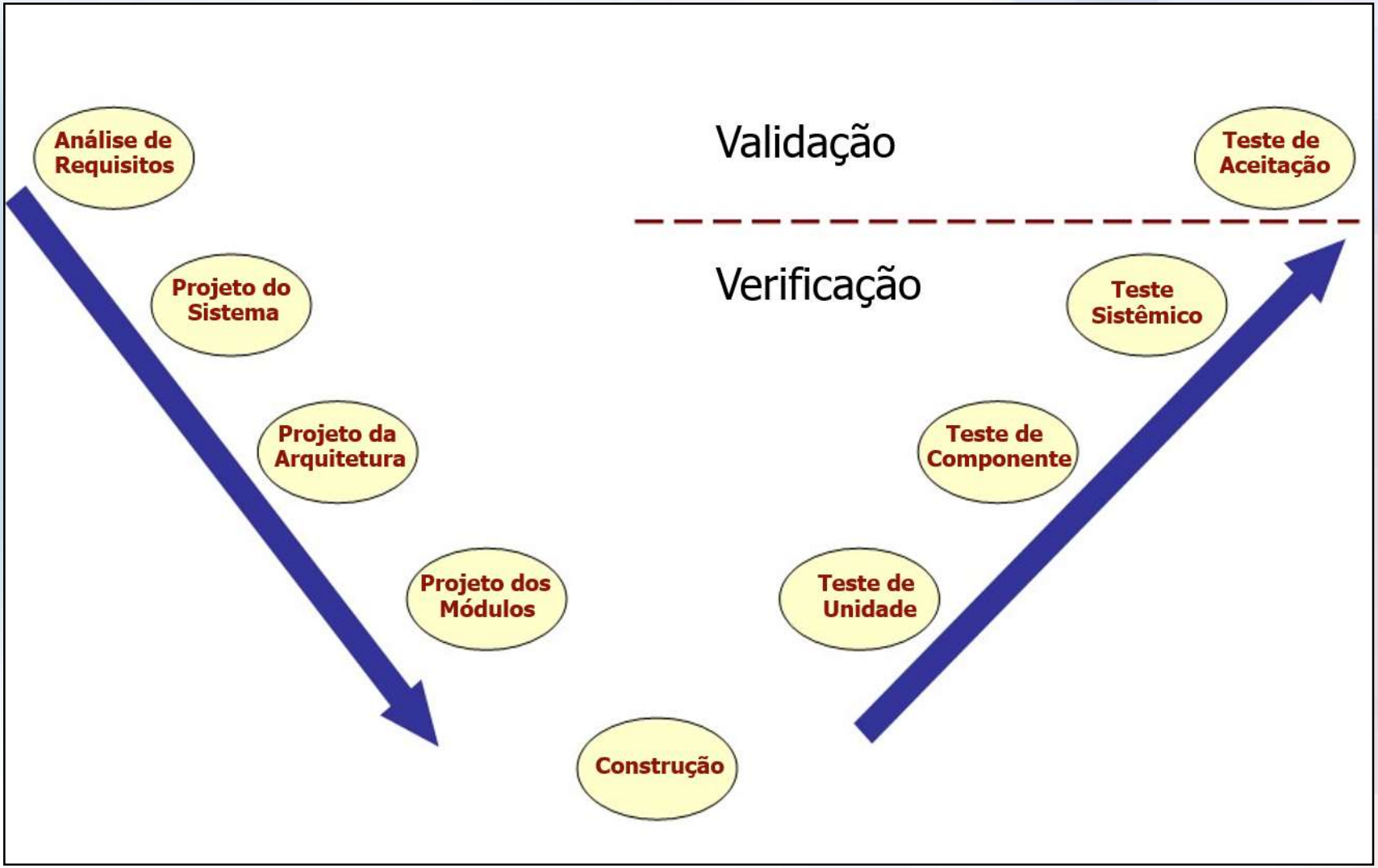
Teste de Software

- Recapitulando: É o processo de executar scripts com o objetivo de encontrar defeitos (Myers, 1979)
- É, portanto, uma atividade de V&V dinâmica
- Do ponto de vista psicológico, o teste de software é uma atividade com um certo viés destrutivo, ao contrário de outras atividades do processo de software

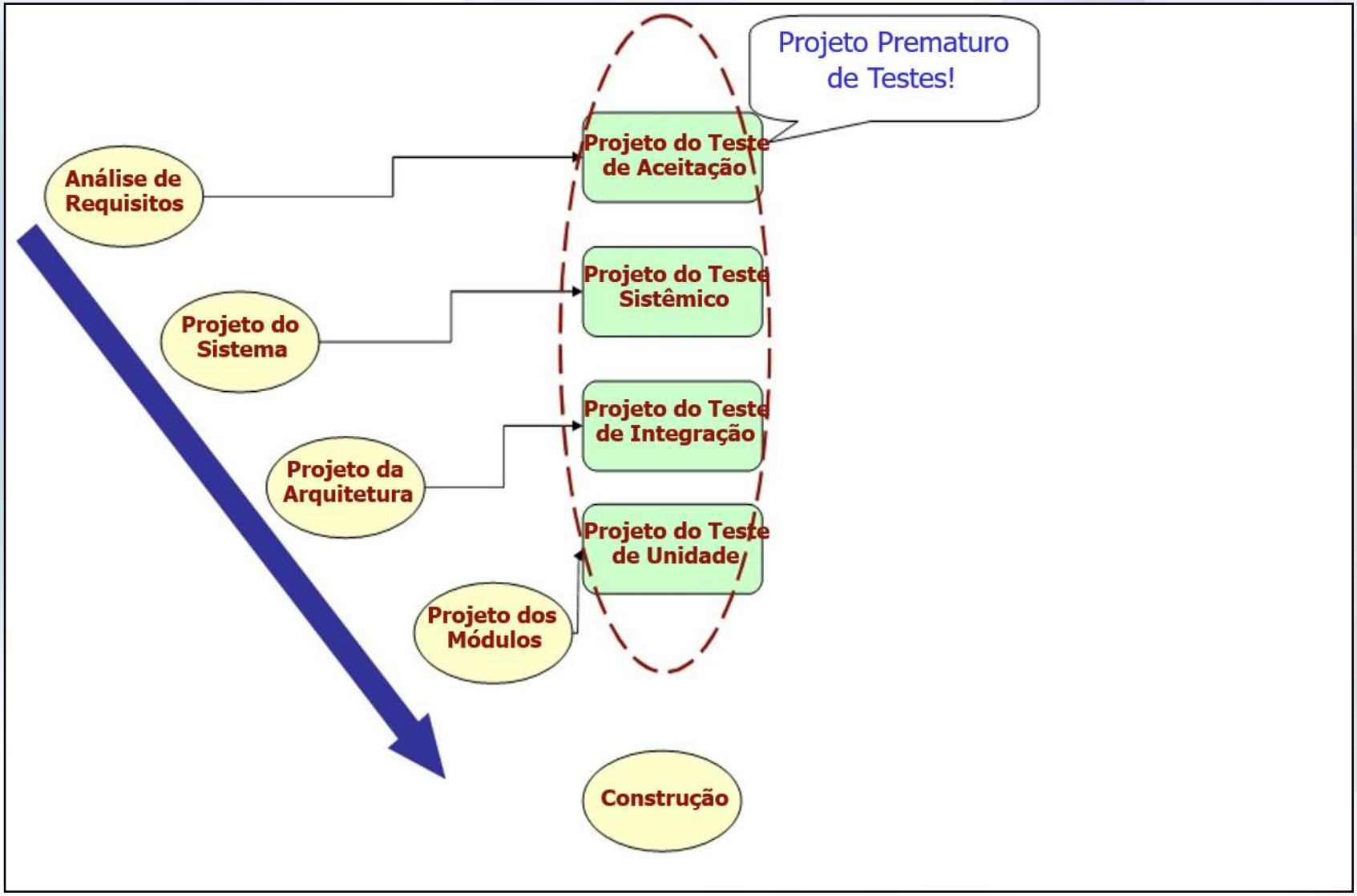
Modelo V (1)


Validação
Verificação
Business Case
Release Testing
Requirements
Acceptance Testing
System Specification
System Testing
System Design
Integration Testing
Unit Design
Unit Testing
Development

Modelo V (2)


Análise de Requisitos
Projeto do Sistema
Projeto da Arquitetura
Projeto dos Módulos
Construção
Validação
Verificação
Teste de Aceitação
Teste Sistêmico
Teste de Componente
Teste de Unidade

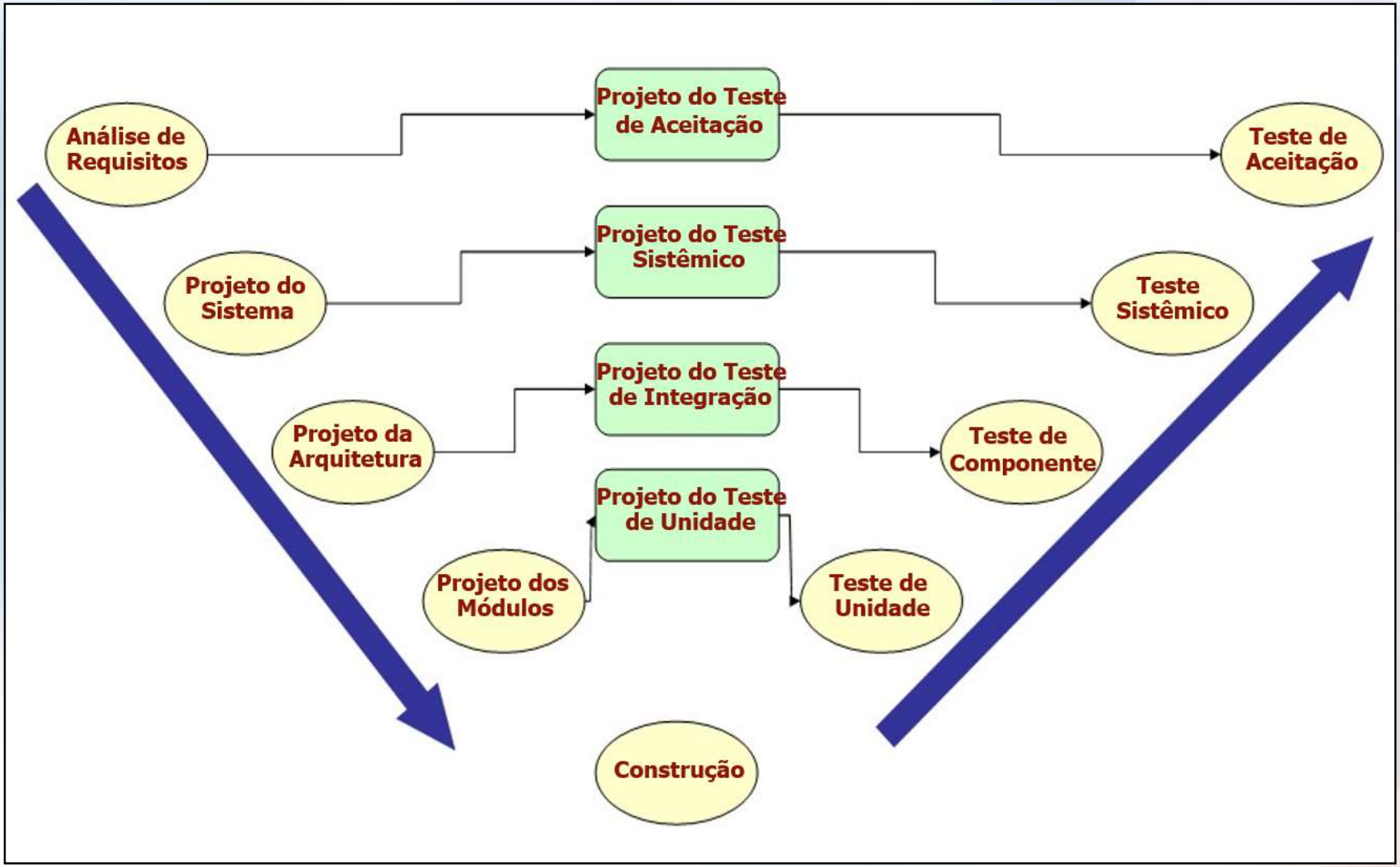
Modelo V (3)


Projeto Prematuro de Testes!
Análise de Requisitos
Projeto do Sistema
Projeto da Arquitetura
Projeto dos Módulos
Projeto do Teste de Aceitação
Projeto do Teste Sistêmico
Projeto do Teste de Integração
Projeto do Teste de Unidade
Construção

Modelo V (4)

- Projeto prematuro dos testes
  - Ao projetar testes, problemas são encontrados
  - Problemas encontrados cedo são mais baratos de corrigir
  - Problemas mais significativos são encontrados primeiro
    - Então que tal verificar logo?
  - Não há trabalho adicional
    - Simplesmente reagende o projeto de testes
  - Projeto de testes pode impactar os requisitos!

Modelo V (5)


Análise de Requisitos
Projeto do Teste de Aceitação
Teste de Aceitação
Projeto do Sistema
Projeto do Teste Sistêmico
Teste Sistêmico
Projeto da Arquitetura
Projeto do Teste de Integração
Teste de Componente
Projeto dos Módulos
Projeto do Teste de Unidade
Teste de Unidade
Construção

Perspectivas de Teste (1)

- Bons **testadores** necessitam de um **conjunto especial** de **habilidades**
  - Um **testador** deve **abordar** um **software** com a **atitude** de **questionar tudo** sobre ele (McGregor e Sykes, 2001)
- A **perspectiva** de **teste** é, um **modo de olhar qualquer produto** de desenvolvimento e **questionar** a sua **validade**
- **Habilidades** requeridas na perspectiva de teste:
  - Querer **prova** de **qualidade**
  - **Não** fazer **suposições**
  - **Não** deixar **passar áreas** importantes
  - Procurar ser **reproduzível**

Perspectivas de Teste (2)

- A **perspectiva** de teste **requer** que um **fragmento** de software **demonstre não apenas** que ele **executa** de acordo com o **especificado**, mas que **executa apenas** o **especificado** (McGregor e Sykes, 2001)

- O **software** faz o que **deveria fazer** e **somente isso**?

- E se **encontrar** mais **coisa** do que deveria ter. **Os Testes identificam isso?**
  - Não, pois o teste trabalha na abordagem V&V, que não contempla características adicionais
  - O Teste valida os requisitos existentes, mas não diz nada sobre os adicionais
  - Mesmo assim, sugere-se reportar sobre funcionalidades adicionais para deixar registrado e evitar problemas futuros

Para Pensar: Exercício

- Como você testaria essa especificação?
  - Um software joga xadrez com um usuário. É exibido o tabuleiro e as peças na tela. Movimentos são feitos arrastando as peças.
  - Proponham 03 Casos de Teste!!
    - Quais foram as metas de aceitação do sistema?
    - Houve necessidade de testes de integração dos módulos??
    - E quanto aos detalhes técnicos dos casos de testes? Muitos específicos? Generalistas?
    - Você verificou e validou também?

Teste de Software

- Executa-se um software ou modelo utilizando algumas entradas em particular e verificar se seu comportamento está de acordo com o esperado
  - Caso a execução apresente algum resultado não especificado, um defeito foi identificado
- Os dados da execução podem servir como fonte para a localização e correção de defeitos, **mas teste não é depuração** (Delamaro et al., 2007).
  - Por quê?

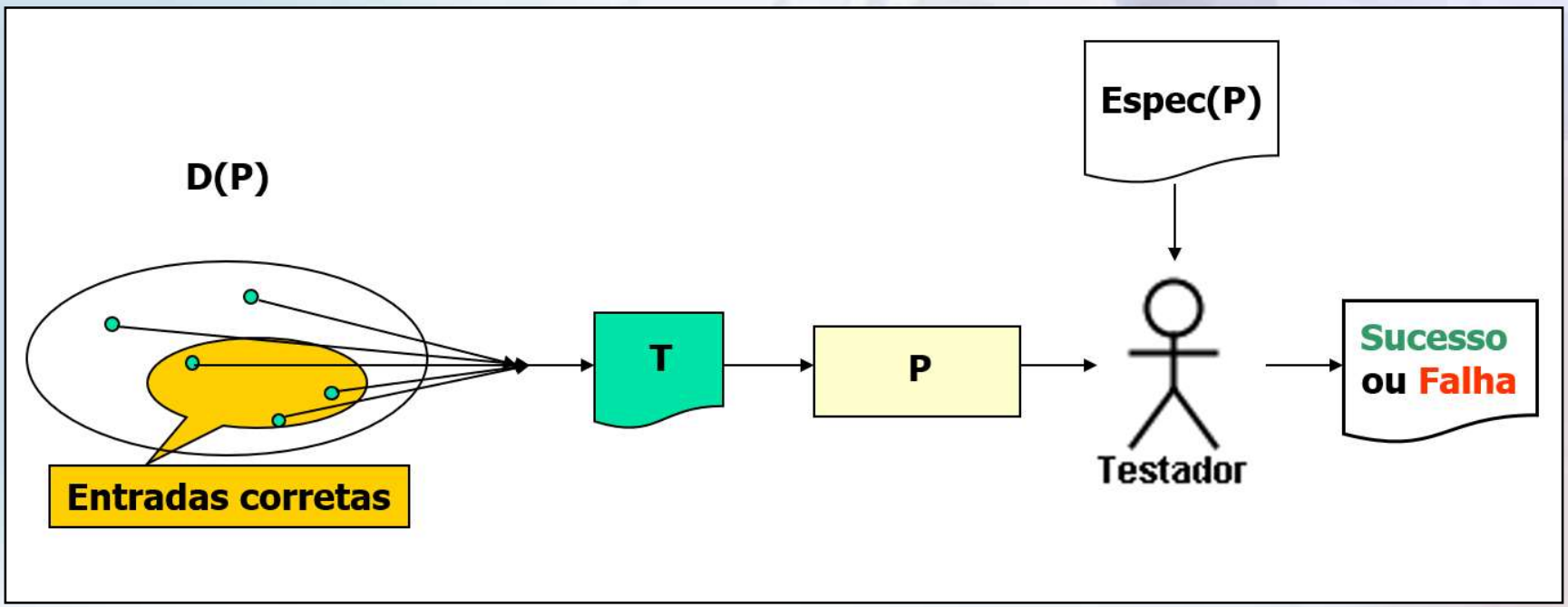
Teste de Software: Conceitos

- Seja **P** um programa a ser testado.
- O domínio de entrada de **P (D(P))** é:
  - o conjunto de todos os valores possíveis que podem ser utilizados para executar **P**
- Um dado de teste para **P** é:
  - um elemento de **D(P)**
- O domínio de saída de **P** é:
  - o conjunto de todos os possíveis resultados produzidos por **P**
- Um **caso de teste** de **P** é:
  - um par formado por um dado de teste mais o resultado esperado para a execução de P com aquele dado de teste.
- Ao conjunto de todos os casos de teste usados durante uma determinada atividade de teste dá-se o nome de conjunto de casos de teste ou conjunto de teste ou **Suíte de Teste** (Delamaro et al., 2007).

Cenário Típico da Atividades de Teste (1)

- Definido uma suíte de testes **T**, executa-se **P** com **T** e verificam-se os resultados obtidos.
- Se os **resultados obtidos coincidem com os resultados esperados**, então nenhum defeito foi identificado (“O software passou no teste”).
- Se, para **algum** caso de teste, o **resultado obtido difere do esperado**, então um defeito foi detectado (“O software não passou no teste”).
- De maneira geral, fica por conta do testador, baseado na especificação do programa, decidir sobre a correção da execução (Delamaro et al., 2007).

Cenário Típico da Atividades de Teste (2)



Teste de Software (1)

- Ao se testar **P**, devem-se selecionar alguns pontos específicos de **D(P)**
- Portanto, testes podem mostrar apenas a presença de defeitos, mas não a ausência deles
- Um aspecto crucial para o sucesso na atividade de teste é a escolha correta dos casos de teste
  - Um teste bem-sucedido identifica defeitos que ainda não foram detectados
- Um **bom caso de teste** é aquele que tem **alta probabilidade de encontrar um defeito** ainda não descoberto

Teste de Software (2)

- A escolha de casos de teste passa pela identificação de **subdomínios de teste**

- Um **subdomínio de teste** é um subconjunto de **D(P)** que contém dados de teste semelhantes, ou seja, que se comportam do mesmo modo; por isso, basta executar P com um deles
  - Fazendo-se isso com todos os subdomínios de **D(P)**, consegue-se um conjunto de teste T bastante reduzido em relação a **D(P)**, mas que, de certa maneira, representa cada um de seus elementos (Delamaro et al., 2007)

Teste de Software (3)

- Existem duas maneiras de se selecionar elementos de cada um dos **subdomínios de teste** (Delamaro et al., 2007):
  - **Teste Aleatório:** um grande número de casos de teste é selecionado aleatoriamente, de modo que, probabilisticamente, se tenha uma boa chance de ter todos os subdomínios representados em **T**
  - **Teste de Subdomínios ou Teste de Partição:** procura-se estabelecer quais são os subdomínios a serem utilizados e, então, selecionam-se os casos de teste em cada subdomínio

Teste de Subdomínios

- A identificação dos subdomínios é feita com base em **critérios de teste**
- Dependendo dos critérios estabelecidos, são obtidos subdomínios diferentes.
- Tipos principais de **critérios de teste**:
  - Funcionais *
  - Estruturais *
  - Baseados em Modelos
  - Baseados em Defeitos *
- O que diferencia cada um deles é o tipo de informação utilizada para estabelecer os subdomínios (Delamaro et al., 2007) → Técnicas de Teste.

Teste Funcional ou Caixa-Preta

- Técnica de projeto de casos de teste na qual o programa ou sistema é considerado uma caixa-preta
- Para testá-lo, são fornecidas entradas e avaliadas as saídas geradas para verificar se estão em conformidade com os objetivos especificados
- Os detalhes de implementação não são considerados
- **O software é avaliado segundo o ponto de vista do usuário** (Delamaro et al., 2007)

Teste Estrutural ou Caixa-Branca

- Estabelece os requisitos de teste com base em uma dada implementação, **requerendo a execução de partes ou componentes elementares de um programa**
- Baseia-se no conhecimento da **estrutura interna** do programa
- **Caminhos lógicos** são testados, estabelecendo casos de teste que põem à prova **condições, laços, definições e usos de variáveis**
- Em geral, a maioria dos critérios estruturais utiliza uma representação de **Grafo de Fluxo de Controle** (Delamaro et al., 2007)

Teste Baseado em Modelos

- Boa parte do tempo despendido na atividade de teste é gasta buscando-se identificar o que o sistema deveria saber
- **Antes de se perguntar se o resultado está correto, deve-se saber qual seria o resultado correto**
- Um modelo é muito útil para tal, servindo tanto como oráculo (instrumento que decide se a saída obtida coincide com a saída esperada) quanto como base para a definição de casos de teste
- Existem diversas técnicas baseadas em Máquinas de Transições de Estados (Delamaro et al., 2007)
  - Tese do ITA: Técnica Baseada no Diagrama de Objetos

Teste Baseado em Defeitos ou Teste de Mutação (1)

- **Defeitos típicos** do processo de **implementação são utilizados como critérios de teste**
- O **programa testado é alterado diversas vezes**, criando-se um conjunto de **programas mutantes**, inserindo defeitos no programa original
- O trabalho do testador é escolher casos de teste que mostrem a diferença de comportamento entre o programa original e os programas mutantes
- Cada mutante determina um subdomínio de teste relacionado com um defeito específico (Delamaro et al., 2007).

Teste Baseado em Defeitos ou Teste de Mutação (2)

- Envolve os seguintes passos:
  1. Geração dos mutantes;
  2. Execução do programa em teste;
  3. Execução dos mutantes;
  4. Análise dos mutantes.

- É praticamente impossível executar 1, 3 e 4 sem apoio computacional.
  - Assim, na prática, deve-se associar à aplicação dessa técnica uma ferramenta de suporte: Proteum (USP)

- Na geração de mutantes, operadores de mutação são aplicados.
  - Ex.: eliminação de comandos, troca de operador relacional, troca de variáveis escalares etc (Delamaro et al., 2007).

Fases de Teste (Estágios de Teste) (1)

- A atividade de teste é dividida em fases com objetivos distintos
- De uma forma geral, pode-se estabelecer como fases (Delamaro et al., 2007):
  - Teste de Unidade
  - Teste de Integração
  - Teste de Sistema

Testes Unitários (1)

- Tem como foco as menores unidades de um programa
- Uma unidade é um componente de software que não pode ser subdividido → Métodos / Funções
- **Nesta fase esperam-se encontrar defeitos relacionados a algoritmos incorretos ou mal implementados, estruturas de dados incorretas ou simples erros de programação**
- Pode ser aplicado à medida que ocorre a implementação das unidades e pode ser realizado pelo próprio desenvolvedor (Delamaro et al., 2007)

Testes Unitários (2)

- Durante os testes de unidade, é necessária a implementação de **drivers** e **stubs**

- Um **driver** é:
  - um programa que **coordena o teste de uma unidade U**, sendo responsável por **ler os dados fornecidos pelo testador**, **repassar** esses **dados** na forma de parâmetros **para U**, **coletar** os **resultados produzidos por U** e **apresentá-los para o testador**

- Um **stub** é:
  - um programa que **substitui, na hora do teste, uma unidade chamada por U**, **simulando** o **comportamento** dessa **unidade** com o **mínimo** de **computação** ou **manipulação** de dados

Testes Unitários (3)

- Todas as técnicas de teste se aplicam ao teste de unidade
- Teste de Mutação é voltado principalmente para o teste de unidade, ainda que haja adaptações para outras fases
  - Por exemplo, mutação de interfaces → teste de integração

Teste de Integração

- Deve ser realizado após serem testadas as unidades individualmente
- **A ênfase é colocada na construção da estrutura do sistema**
- Deve-se **verificar se as partes**, quando colocadas para trabalhar **juntas, não conduzem a erros**
- Requer grande conhecimento das estruturas internas do sistema e, por isso, geralmente é executado pela própria equipe de desenvolvimento (Delamaro et al., 2007)
- Todas as técnicas de teste se aplicam, com destaque para o teste funcional

Teste de Sistemas

- Uma vez integradas todas as partes, inicia-se o teste de sistema
- Quando realizado por uma equipe de teste, o objetivo é:
  - Validar se as funcionalidades especificadas na fase de requisitos foram corretamente implementadas
- Quando realizado por usuários, o objetivo é:
  - Validar o sistema (Teste de Aceitação)
- É uma boa prática que essa fase seja realizada por testadores independentes
- Tipicamente, aplica-se teste funcional

Engenharia de Software II / Qualidade e Teste de Software

Aula 06: V&V: Verificação e Validação, Modelo em V e Definição Formal de Testes

Breno Lisi Romano

Dúvidas?

http://sites.google.com/site/blromano

Instituto Federal de São Paulo – IFSP São João da Boa Vista

Bacharelado em Ciência da Computação – BCC (ENSC6)
Tecnologia em Sistemas para Internet – TSI (QTSI6)

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista