SSC5877 – Validação Verificação e Teste de Software

Introdução

Prof. Marcio E. Delamaro

delamaro@icmc.usp.br

Motivação

- Estudos mostram
  - Praticamente todos os “módulos” de um software possuem defeitos durante o desenvolvimento
  - Após a liberação 40% estão livres de defeitos
  - É praticamente impossível liberar software sem defeito

Motivação

- Estudos mostram
  - Praticamente todos os “módulos” de um software possuem defeitos durante o desenvolvimento
  - Após a liberação 40% estão livres de defeitos
  - É praticamente impossível liberar software sem defeito
- Interesse crescente na atividade de teste principalmente pela crescente demanda por software de alta qualidade

Motivação

- Estudos mostram
  - Praticamente todos os “módulos” de um software possuem defeitos durante o desenvolvimento
  - Após a liberação 40% estão livres de defeitos
  - É praticamente impossível liberar software sem defeito
- Interesse crescente na atividade de teste principalmente pela crescente demanda por software de alta qualidade
- Grupos de pesquisa em teste de software proliferam no mundo

Motivação

- Estudos mostram
  - Praticamente todos os “módulos” de um software possuem defeitos durante o desenvolvimento
  - Após a liberação 40% estão livres de defeitos
  - É praticamente impossível liberar software sem defeito
- Interesse crescente na atividade de teste principalmente pela crescente demanda por software de alta qualidade
- Grupos de pesquisa em teste de software proliferam no mundo
- Vagas na indústria para engenheiros de teste se multiplicam

O que é teste?

- Forma de procurar garantir que o produto de software atende aos requisitos que foram definidos para ele.
  - **Correção**, desempenho, segurança, etc

O que é teste?

- Forma de procurar garantir que o produto de software atende aos requisitos que foram definidos para ele.
  - **Correção**, desempenho, segurança, etc
- Não é a única forma mas talvez a mais importante

O que é teste?

- Forma de procurar garantir que o produto de software atende aos requisitos que foram definidos para ele.
  - **Correção**, desempenho, segurança, etc
- Não é a única forma mas talvez a mais importante
- De acordo com IEEE Standard Glossary 610.12-1990 (R 2002)
  - *The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component*

Engenheiros de software são abençoados

- Ver filme

Engenheiros de software são abençoados

- Ver filme
- Temos a chance de testar nosso produto antes que ele seja usado

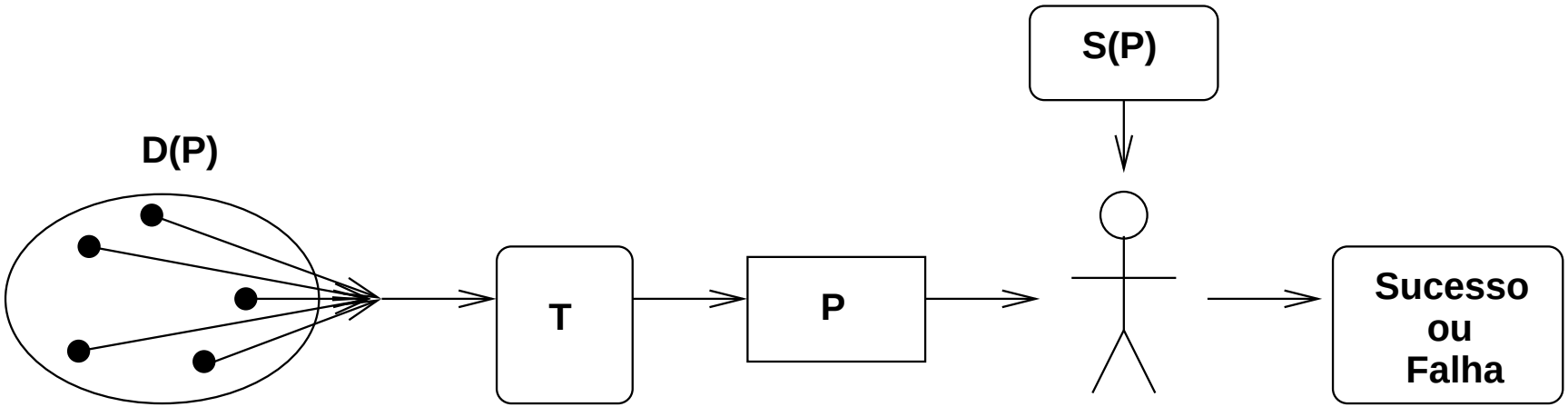
Engenheiros de software são abençoados

- Ver filme
- Temos a chance de testar nosso produto antes que ele seja usado
- Basta fazê-lo direito

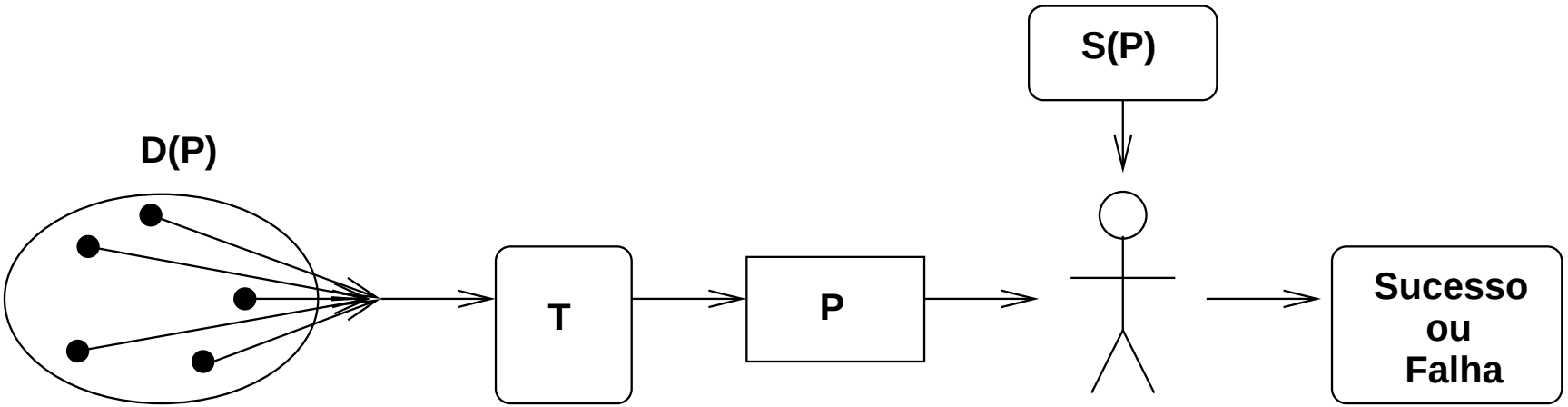
Engenheiros de software são abençoados

- Ver filme
- Temos a chance de testar nosso produto antes que ele seja usado
- Basta fazê-lo direito
- O que não é tão simples assim
  - Therac 25
  - Ariane 5
  - Pathfinder rover
  - Satélite Phobos II
  - Airbus 320 – Varsóvia

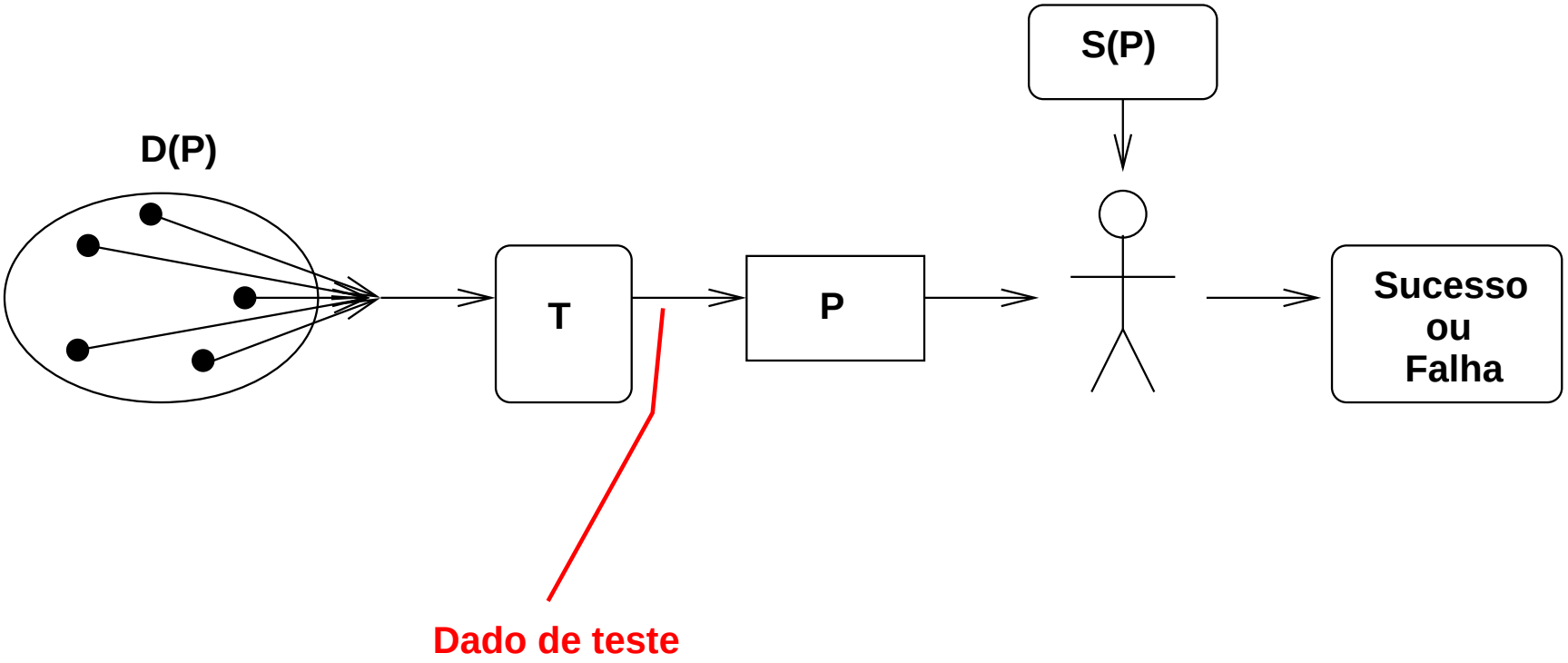
O que é teste



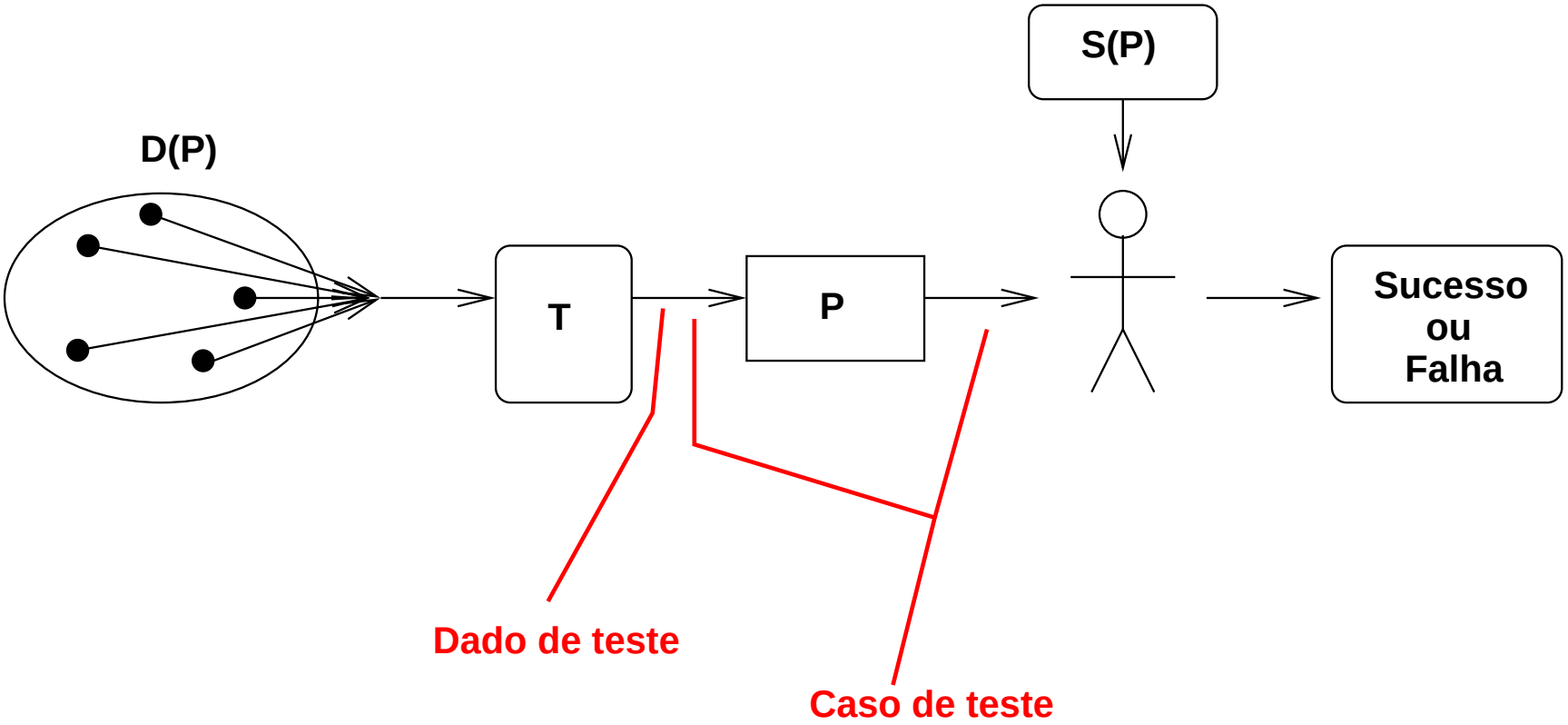
Algumas definições



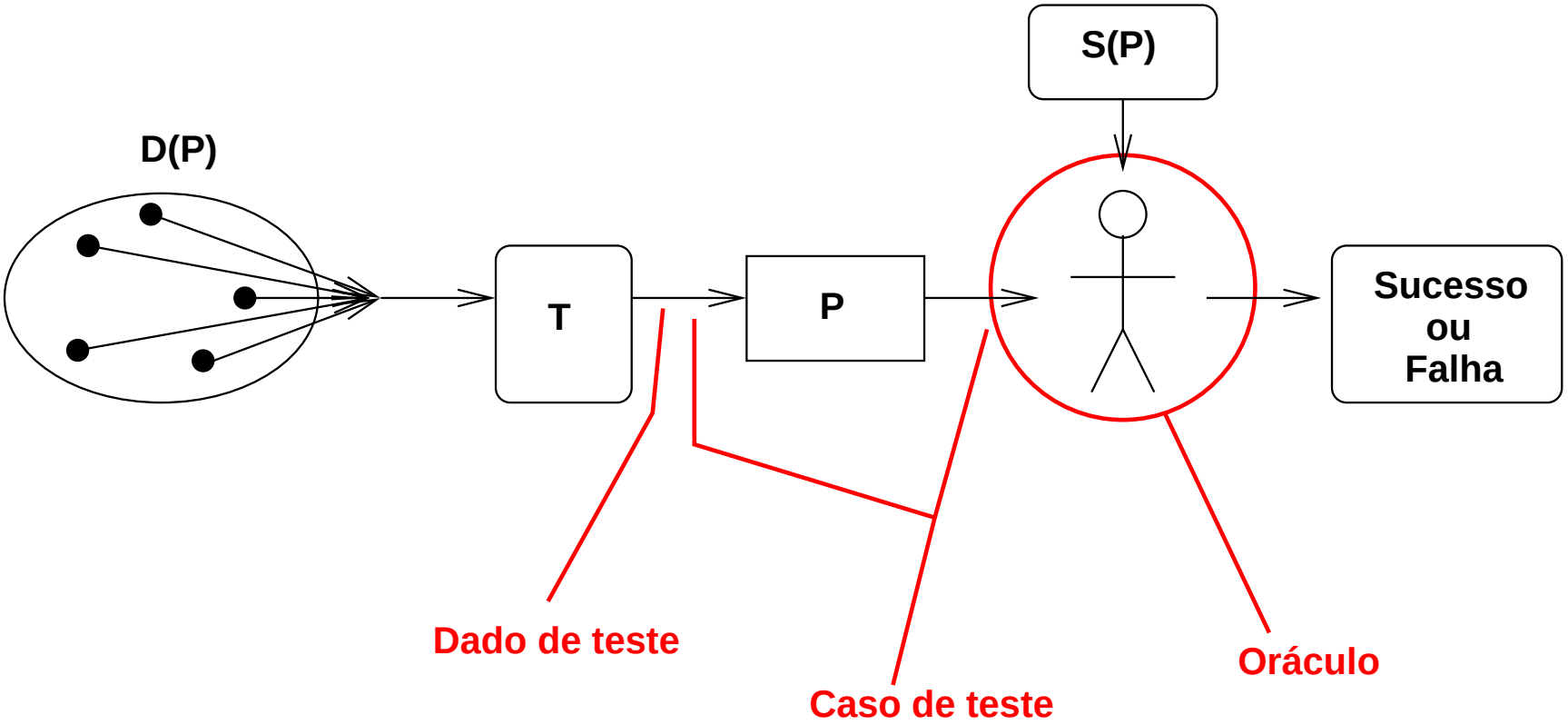
Algumas definições



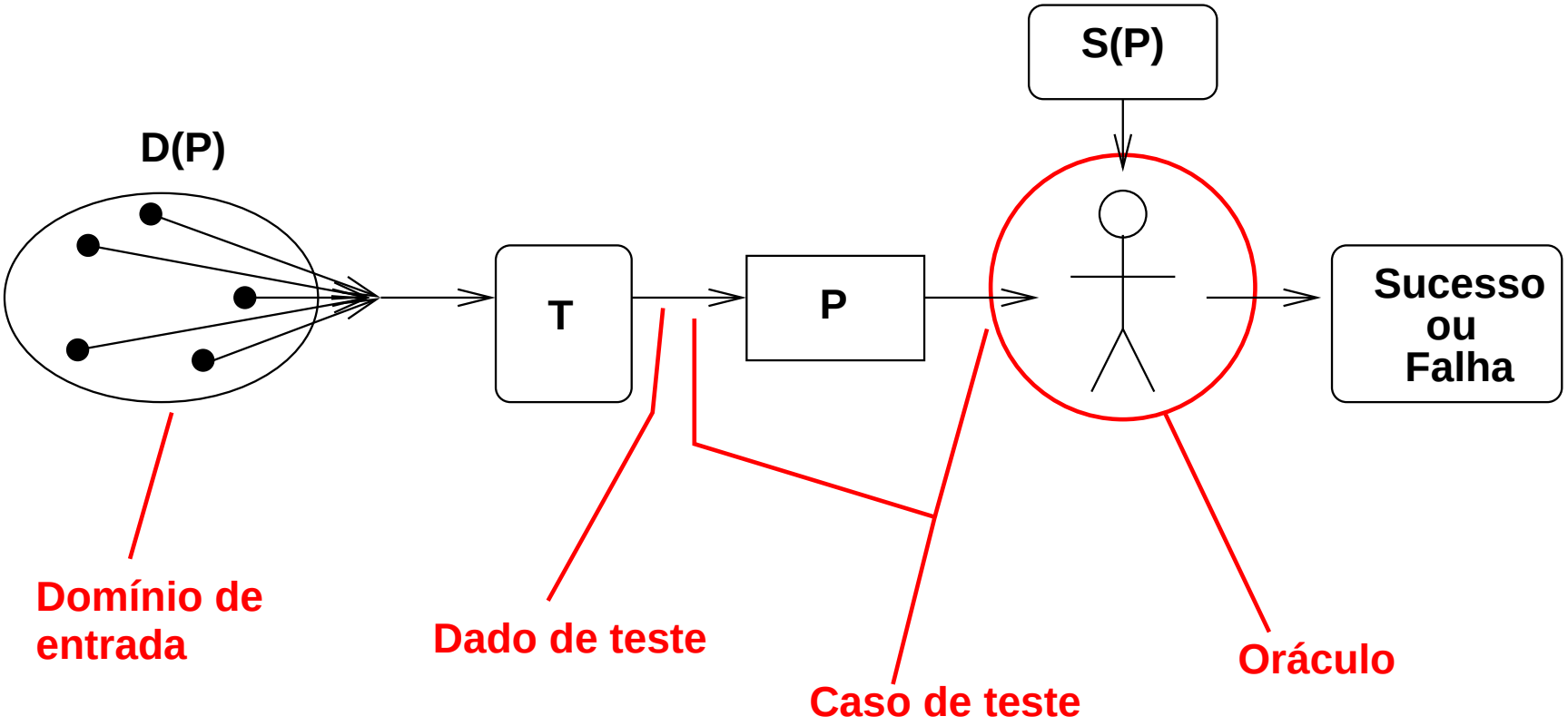
Algumas definições



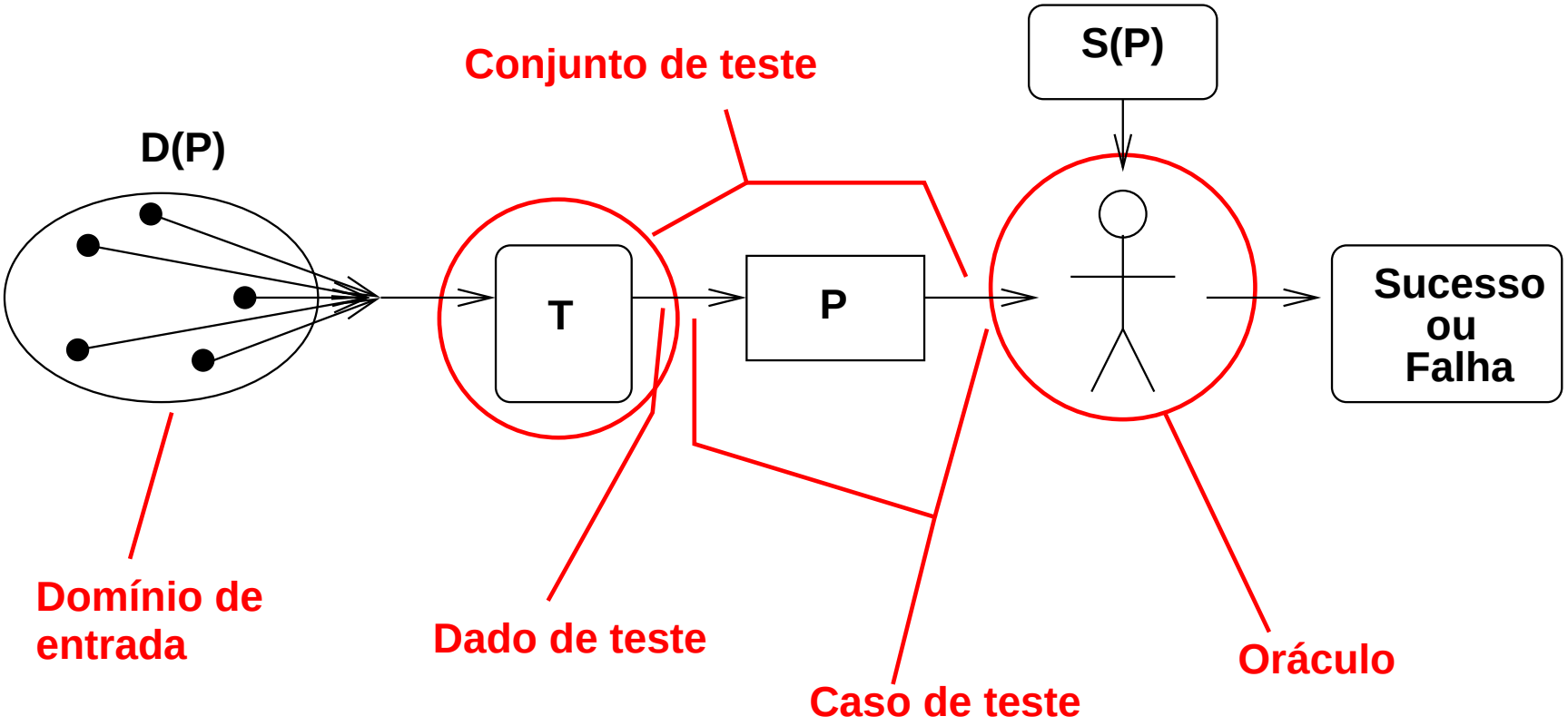
Algumas definições



Algumas definições



Algumas definições



Desafios do Teste

- Alguém já testou algum produto de software?
- Quais foram os maiores desafios?
- Alguns problemas comuns são:
  - Não há tempo para o teste exaustivo.
  - Muitas combinações de entrada para serem exercitadas.
  - Dificuldade em determinar os resultados esperados para cada caso de teste.
  - Requisitos do software inexistentes ou que mudam rapidamente.
  - Não há tempo suficiente para o teste.
  - Não há treinamento no processo de teste.
  - Não há ferramenta de apoio.
  - Gerentes que desconhecem teste ou que não se preocupam com qualidade.

Partes de um Caso de Teste

- Um caso de teste bem projetado é dividido em três partes:
  - Entradas (*inputs*);
  - Saídas Esperadas (*outputs*);
  - Ordem de execução (*order of execution*).

Entradas

- Geralmente identificadas como dados fornecidos via teclado para o programa executar.
- Entretanto, os dados de entrada podem ser fornecidos por outros meios, tais como:
  - Dados oriundos de outro sistema que servem de entrada para o programa em teste;
  - Dados fornecidos por outro dispositivo;
  - Dados lidos de arquivos ou banco de dados;
  - O estado do sistema quando os dados são recebidos;
  - O ambiente no qual o programa está executando.

Saídas

- As saídas também podem ser produzidas de diferentes formas.
- A mais comum é aquele apresentada na tela do computador.
- Além dessa, as saídas podem ser enviadas para:
  - Outro sistema interagindo com o programa em teste;
  - Dados enviados para um dispositivo externo;
  - Dados escritos em arquivos ou banco de dados;
  - O estado do sistema ou o ambiente de execução podem ser alterados durante a execução do programa.

Oráculo

- Todas as formas de entrada e saída são relevantes.
- Durante o projeto de um caso de teste, determinar correção da saída esperada é função do **oráculo** (`oracle`).
- Oráculo corresponde a um mecanismo (programa, processo ou dados) que indica ao projetista de casos de testes a saída obtida para um caso de teste é aceitável ou não.

Oráculo (2)

- Beizer (1990) lista cinco tipos de oráculos:
  - Oráculo “Kiddie” - simplesmente execute o programa e observe sua saída. Se ela parecer correta deve estar correta.
  - Conjunto de Teste de Regressão - execute o programa e compare a saída obtida com a saída produzida por uma versão mais antiga do programa.
  - Validação de Dados - execute o programa e compare a saída obtida com uma saída padrão determinada por uma tabela, fórmula ou outra definição aceitável de saída válida.
  - Conjunto de Teste Padrão - execute o programa com um conjunto de teste padrão que tenha sido previamente criado e validado. Utilizado na validação de compiladores, navegadores Web e processadores de SQL.
  - Programa existente - execute o programa em teste e o programa existente com o mesmo caso de teste e compare as saídas. Semelhante ao teste de regressão.

Ordem de Execução

- Existem dois estilos de projeto de casos de teste relacionados com a ordem de execução:
  - Casos de teste em cascata.
  - Casos de teste independentes.

Ordem de execução (2)

- Casos de teste em cascata - quando os casos de teste devem ser executados um após o outro, em uma ordem específica. O estado do sistema deixado pelo primeiro caso de teste é reaproveitado pelo segundo e assim sucessivamente.
  Por exemplo, considere o teste de uma base de dados:
  - criar um registro
  - ler um registro
  - atualizar um registro
  - ler um registro
  - apagar um registro
  - ler o registro apagado

Ordem de execução (3)

- Vantagem - casos de testes tendem a ser pequenos e simples. Fáceis de serem projetados, criados e mantidos.
- Desvantagem - se um caso de teste falhar os casos de teste subseqüentes também podem falhar.

Ordem de execução (4)

- Casos de teste independentes - Cada caso de teste é inteiramente autocontido.
  - Vantagem - casos de teste podem ser executados em qualquer ordem.
  - Desvantagem - casos de teste tendem a ser grandes e complexos, mais difíceis de serem projetados, criados e mantidos.

Tipos de Teste (1)

- Diferentes tipos de testes podem ser utilizados para verificar se um programa se comporta como o especificado.
- Basicamente, os testes podem ser classificados em teste funcional, teste estrutural ou teste baseado em defeito.
- Esses **tipos de teste** correspondem às chamadas **técnicas de teste**.

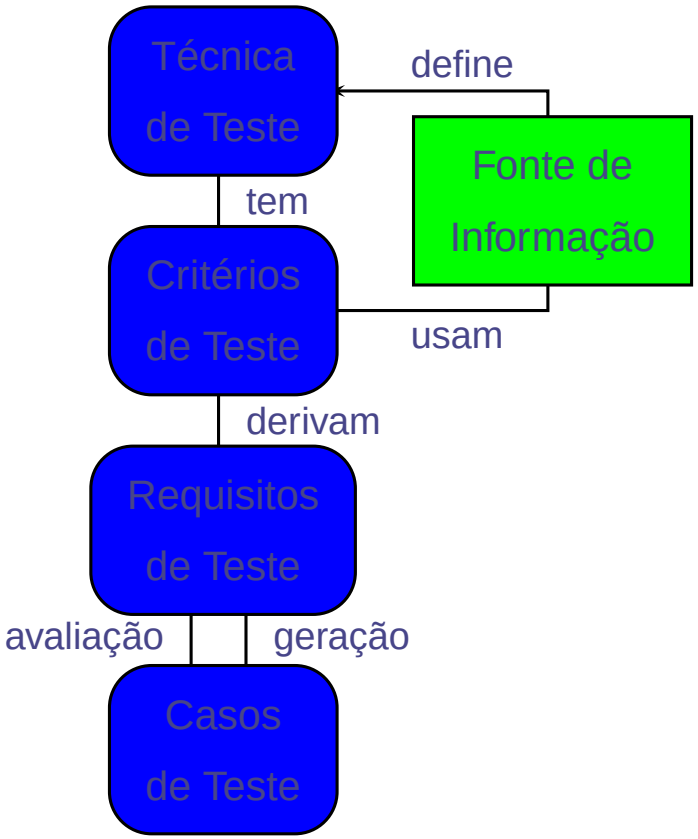
Tipos de Teste (2)

- A técnica de teste é definida pelo tipo de informação utilizada para realizar o teste.
  - **Técnica funcional** - os testes são baseados exclusivamente na especificação de requisitos do programa. Nenhum conhecimento de como o programa está implementado é requerido.
  - **Técnica estrutural** - os testes são baseados na estrutura interna do programa, ou seja, na implementação do mesmo.
  - **Técnica baseada em defeito** - os testes são baseados em informações histórica sobre defeitos cometidos freqüentemente durante o processo de desenvolvimento de software.

Tipos de Teste (3)

- Cada técnica de teste possui um conjunto de **critérios de teste**.
- Os critérios sistematizam a forma como **requisitos de teste** devem ser produzidos a partir da fonte de informação disponível (especificação de requisitos, código fonte, histórico de defeitos, dentre outras).
- Os requisitos de teste são utilizados para:
  - Gerar casos de teste.
  - Avaliar a qualidade de um conjunto de teste existente.

Técnicas, Critérios e Requisitos de Teste

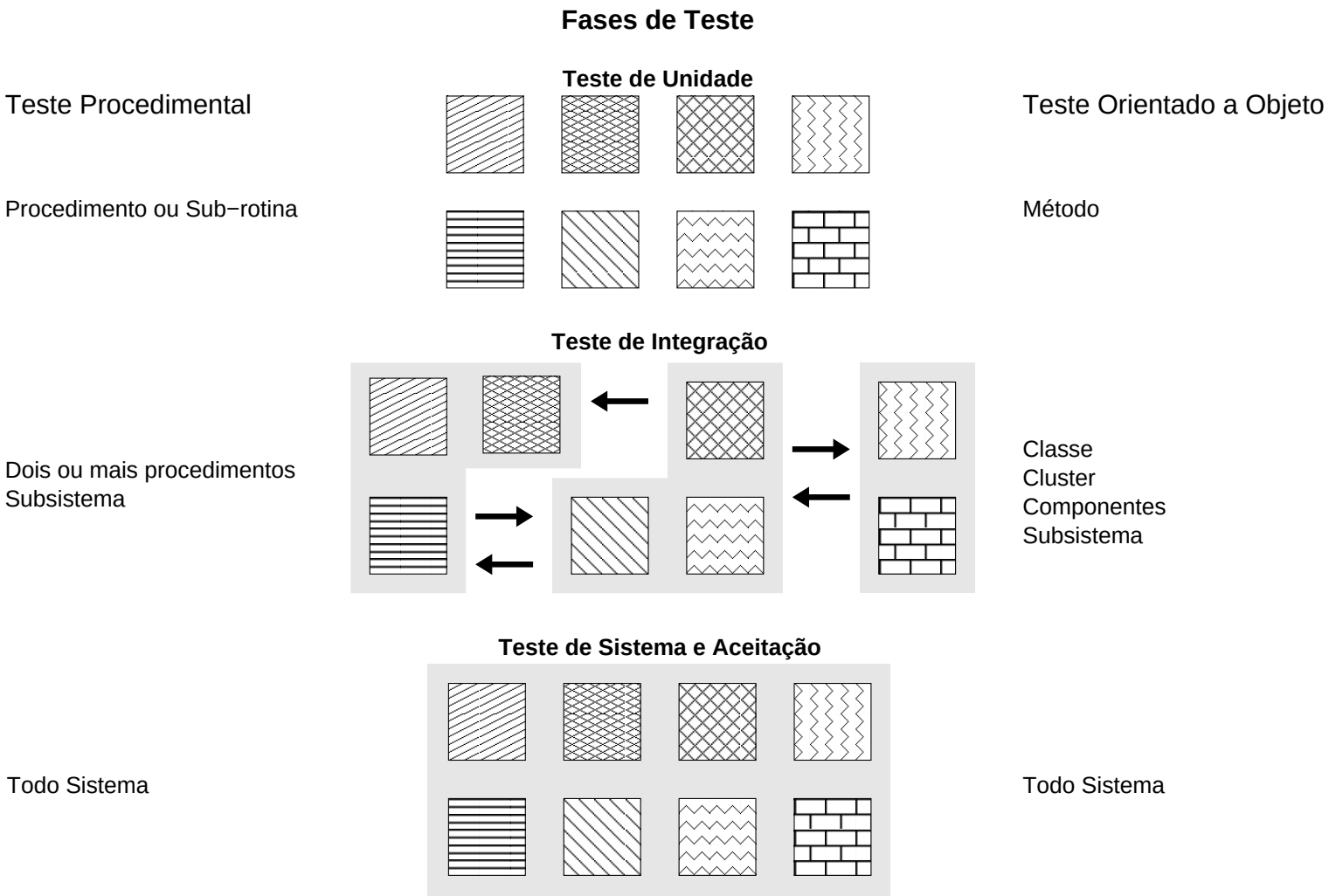


Termos utilizados na área de teste e seus relacionamentos

Fases de teste (1)

- A atividade de teste também é dividida em fases, conforme outras atividades de Engenharia de Software.
- Objetivo é reduzir a complexidade dos testes.
- Conceito de “dividir e conquistar”.
- Começar testar a menor unidade executável até atingir o programa como um todo.

Fases de Teste (2)



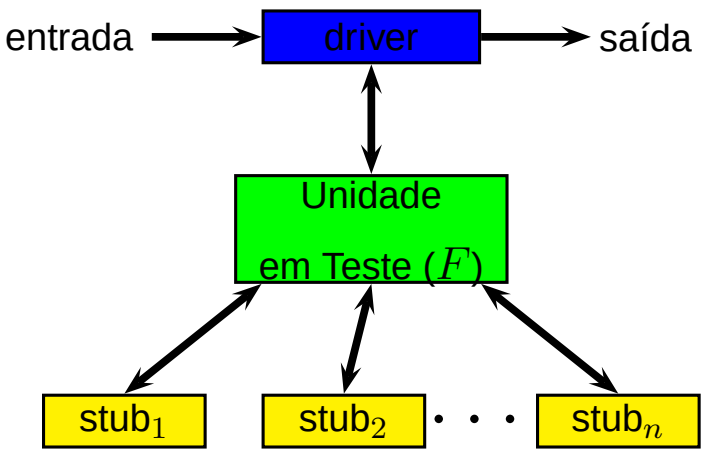
Principais Fases de Teste (adaptada de Binder (1999)).

Teste de Unidade

- Objetivo é identificar erros de lógica e de programação na menor unidade de programação.
- Diferentes linguagens possuem unidades diferentes.
  - Pascal e C possuem procedimentos ou funções.
  - Java e C++ possuem métodos (ou classes?).
  - Basic e COBOL (o que seriam unidades?).
- Como testar uma unidade que depende de outra para ser executada?
- Como testar uma unidade que precisa receber dados de outra unidade para ser executada?

Driver e *Stub*

- Para auxiliar no teste de unidade, em geral, são necessários *drivers* e *stubs*.
- O *driver* é responsável por fornecer para uma dada unidade os dados necessários para ela ser executada e, posteriormente, apresentar os resultados ao testador.
- O *stub* serve para simular o comportamento de uma unidade que ainda não foi desenvolvida, mas da qual a unidade em teste depende.



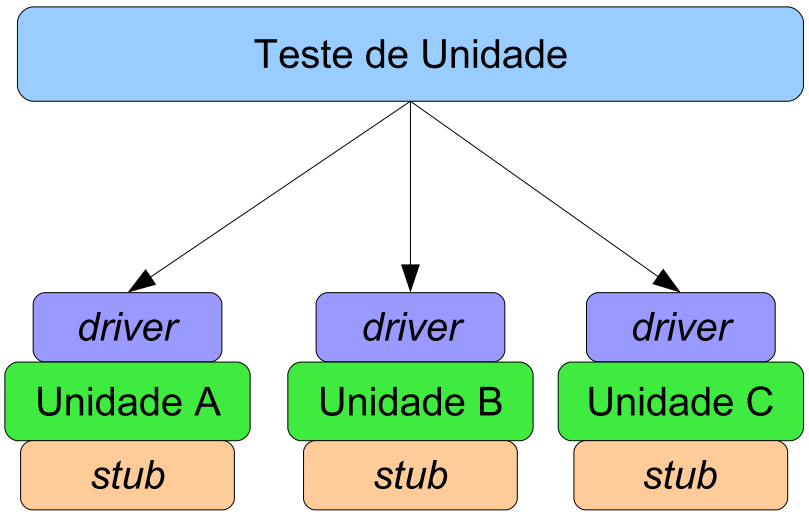
Drivers e *Stubs*.

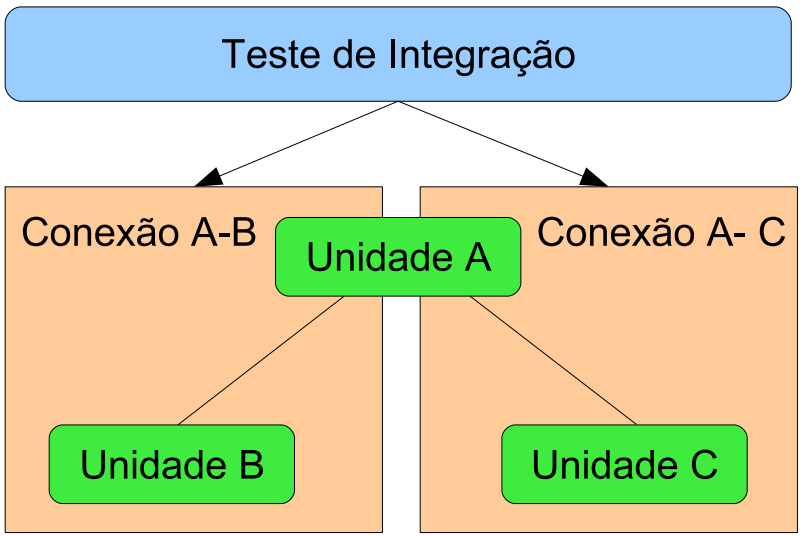
Teste de Integração (1)

- Objetivo é verificar se as unidades testadas individualmente se comunicam como desejado.
- Por que testar a integração entre unidades se as mesmas, em isolado, funcionam corretamente?

Teste de Integração (2)

- Dados podem se perder na interface das unidades.
- Variáveis globais podem sofrer alterações indesejadas.





Teste de Unidade X Teste de Integração.

Teste de Sistema

- Objetivo é verificar se o programa em si interage corretamente com o sistema para o qual foi projetado. Isso inclui, por exemplo, o SO, banco de dados, hardware, manual do usuário, treinamento, etc.
- Corresponde a um teste de integração de mais alto nível.
- Inclui teste de funcionalidade, usabilidade, segurança, confiabilidade, disponibilidade, performance, backup/restauração, portabilidade, entre outros (Norma ISO-IEC-9126 para mais informações ISO/IEC (1991))

Teste de Aceitação

- Objetivo é verificar se o programa desenvolvido atende as exigências do usuário.

Teste de Regressão

- Mesmo após liberado o software precisa ser testado
- A cada manutenção é preciso verificar se o software mantém suas características
- Se nenhum efeito colateral foi introduzido
- Técnicas de teste de regressão reutilizam subconjuntos do conjunto de teste existente

Limitações do Teste (1)

Observe o exemplo abaixo extraído de Binder (1999):

```
1 int blech(int j) {
2     j = j - 1;  // deveria ser j = j + 1
3     j = j / 30000;
4     return j;
5 }
```

Código fonte: `./src/blech.java`

- Considerando o tipo inteiro com 16 bits (2 bytes) – o menor valor possível seria -32.768 e o maior seria 32.767, resultando em 65.536 valores diferentes possíveis.
- Haverá tempo suficiente para se criar 65.536 casos de teste? E se os programas forem maiores? Quantos casos de teste serão necessários?

Limitações do Teste (2)

```
1 int blech(int j) {
2     j = j - 1;  // deveria ser j = j + 1
3     j = j / 30000;
4     return j;
5 }
```

Código fonte: `./src/blech.java`

Quais valores escolher?

Entrada (j)	Saída Esperada	Saída Obtida
1	0	0
42	0	0
40000	1	1
-64000	-2	-2

Quais valores de entrada revelam o erro no programa acima?

Limitações do Teste (3)

- Os casos de testes anteriores não revelam o erro.
- Somente quatro valores do intervalo de entrada válido revelam o erro:
- Os valores abaixo revelam o erro:

Entrada (j)	Saída Esperada	Saída Obtida
-30000	0	-1
-29999	0	-1
30000	1	0
29999	1	0

- Qual a chance desses valores serem selecionados???

Automatização

- A atividade de teste é cara e consome muito tempo.
- Automatizar significa:
  - Diminuir custos
  - Diminuir erros
  - Facilitar teste de regressão
- Atividades intelectuais $\times$ atividades braçais.
- Suporte a técnicas de teste
- Geração de dados de teste
- Oráculos

Algumas definições

- Validação: o processo de avaliar o software, ao final do seu desenvolvimento com o objetivo de garantir conformidade com o seu uso esperado.
  - Requer conhecimento do domínio para o qual o software foi desenvolvido
- Verificação: o processo de determinar se os produtos de uma determinada fase de desenvolvimento do software estão de acordo com os requisitos estabelecidos nas fases anteriores.
  - Atividade mais técnica, que usa conhecimento sobre artefatos particulares, requisitos e especificações.
- Teste: Avaliação do software por meio da observação de sua execução.

Erros, etc

- Defeito: uma incorreção estática no código.
- Erro: um estado interno incorreto, que é consequência da execução de um defeito.
- Falha: comportamento externo incorreto, com respeito aos requisitos ou qualquer outra descrição do comportamento esperado.

Uma analogia Ammann e Offut (2008)

- Considere um médico fazendo um diagnóstico num paciente doente. O paciente entra no consultório com uma lista de “falha” (ou seja, os sintomas). O médico tem que descobrir o “defeito”, ou seja, a raiz, a causa do problema. Para isso, o médico pode solicitar exames exames que vão procurar condições internas anômalas como pressão alta, batida irregular, colesterol, etc. Essas condições anômalas, na nossa terminologia seriam os “erros”.
- Defeitos de software tem origem num “engano” humano.

Exemplo

```
1  public static int numZero (int[] x) {
2  // Funcao: se x == null lanca NullPointerException
3  // senao returna o número de ocorrências de 0 em x
4     int count = 0;
5     for (int i = 1; i < x.length; i++)
6     {
7        if (x[i] == 0)
8        {
9           count++;
10       }
11    }
12    return count;
13 }
```

Código fonte: `./src/defeito.java`

Exemplo

- Defeito: inicialização da variável $i = 1$
- Não falha: $x = [2, 7, 0]$
- Falha: $x = [0, 7, 2]$
- Estado dado pelo valor de: x, i, count e PC
- Erro: $x = [0, 7, 2], i = 1, count = 0$, PC = if
- Erro (???): $x = [2, 7, 0], i = 1, count = 0$, PC = if

Para uma falha ocorrer (RIP)

- O ponto do programa que contém um defeito deve ser executado (alcançabilidade).
- Após a execução deste ponto, o estado da execução deve ser incorreto (infecção).
- O estado infectado deve se propagar de modo a produzir uma saída incorreta (propagação).
- RIP (Reachability, Infection, Propagation).

Exercícios

```
1  public static int lastZero (int[] x) {
2  // Funcao: se x==null lanca NullPointerException
3  // senao returna o indic do ultimo 0 em x
4  // Returna -1 se 0 nao ocorre em x
5
6     for (int i = 0; i < x.length; i++) {
7        if (x[i] == 0) {
8           return i;
9        }
10    }
11    return -1;
12 }
13 // teste: x=[0, 1, 0]
14 // esperado = 2
```

Código fonte: `src/lastzero.java`

lastZero

- a) Identifique o defeito
- b) Se possível, identifique um caso de teste que não executa o defeito.
- c) Se possível, identifique um caso de teste que executa o defeito mas não resulta em um estado de erro.
- d) Se possível, identifique um caso de teste que resulta em um estado de erro mas não em uma falha.
- e) Para o caso de teste dado, identifique o primeiro estado de erro. Descreva o estado por completo.
- f) Corrija o defeito e verifique que o caso de teste dado produz o resultado correto.

oddOrPos

```
1  public static int oddOrPos(int[] x) {
2  // Funcao: se x==null lanca NullPointerException
3  // senao returna o numero de elementos em x que
4  // sao impar ou positivo (ou ambos)
5     int count = 0;
6     for (int i = 0; i < x.length; i++) {
7        if (x[i]%2 == 1 || x[i] > 0) {
8           count++;
9        }
10    }
11    return count;
12 }
13 // test: x=[-3, -2, 0, 1, 4]
14 // Expected = 3
```

Código fonte: `src/oddorpos.java`

Referências

AMMANN, P.; OFFUT, J. *Introduction to software testing*. 1a. ed. Cambridge University Press, 2008.

BEIZER, B. *Software testing techniques*. 2nd ed. New York: Van Nostrand Reinhold Company, 1990.

BINDER, R. V. *Testing object-oriented systems: Models, patterns, and tools*, v. 1. Addison Wesley Longman, Inc., 1999.

ISO/IEC *Quality characteristics and guidelines for their use*. Padrão ISO/IEC 9126, ISO/IEC, 1991.