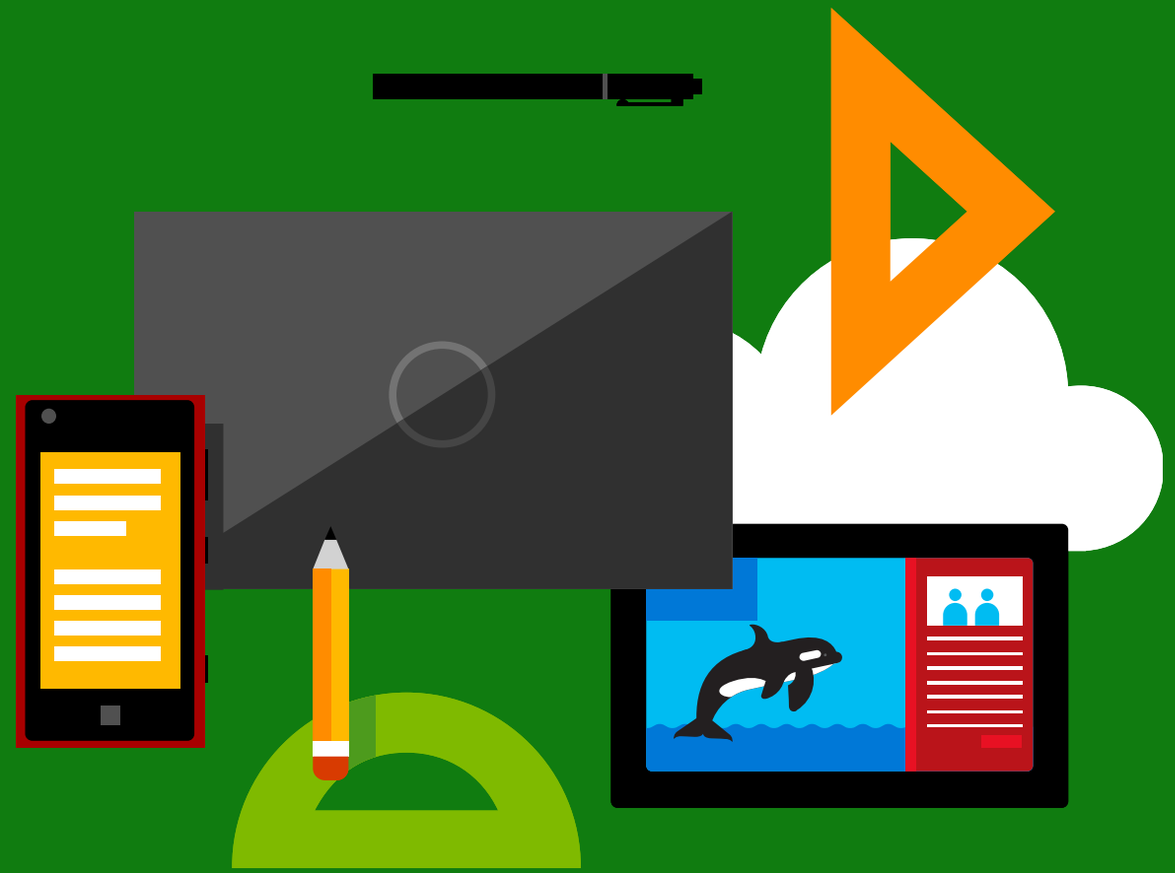
Desenvolvimento Web

Trilha JavaScript com Angular e Node

Instrutor: Júlio Pereira Machado (julio.machado@pucrs.br)

Teste de Software



Técnicas de Teste

- Envolvem executar uma implementação do software com dados de teste e examinar suas saídas e seu comportamento operacional
- Tipos de testes:
  - Os estatísticos
    - São utilizados para testar o desempenho e a confiabilidade do programa e como ele se comporta sob condições operacionais
  - Os de defeitos
    - Se destinam a encontrar inconsistências entre o programa e sua especificação
    - Podem ser usados tanto para verificação como para validação

Falhas, Erros e Defeitos

- **Defeito (*failure*):** é a manifestação externa do estado de erro quando este interfere na saída gerada
- **Erro (*error*):** é o estado interno que diverge do estado correto
- **Falha (*fault*):** problema no código que, ao ser executado, leva a um erro

- A execução de um teste leva a detecção de defeitos. A depuração do sistema permite então detectar o estado de erro e levar, possivelmente, a descoberta da falha.

Exemplo

- Um programa deve imprimir um cupom fiscal com o valor total da compra
- Executando-se um caso de teste percebe-se um defeito: o valor da soma está incorreto
- A depuração mostra que o preço do último produto não está sendo somado: estado de erro
- Detecta-se a falha: o método que percorre a lista de produtos está interpretando erroneamente o tamanho da lista ... (size-1)

Exemplo

Falha: deve iniciar busca em 0, não 1

```
public static int numZero (int [ ] arr)
{  // Efeito: se arr é null throw NullPointerException
   // senão retornar número ocorrências de 0 em arr
   int count = 0;
   for (int i = 1; i < arr.length; i++)
   {
      if (arr [ i ] == 0)
      {
         count++;
      }
   }
   return count;
}
```

Teste 1
[2, 7, 0]
Esperado: 1
Atual: 1

Erro: i é 1, não 0, na primeira iteração
Defeito: nenhum

Teste 2
[0, 2, 7]
Esperado: 1
Atual: 0

Erro: i é 1, não 0
Erro se propaga para a variável count
Defeito: count é 0 no comando return

O Termo Bug

- *Bug* é utilizado informalmente

“It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise—this thing gives out and *[it is]* then that 'Bugs'—as such little faults and difficulties are called—show themselves and months of intense watching, study and labor are requisite. . .” – Thomas Edison

“an analyzing process must equally have been performed in order to furnish the Analytical Engine with the necessary operative data; and that herein may also lie a possible source of error. Granted that the actual mechanism is unerring in its processes, the cards may give it wrong orders.” – Ada, Countess Lovelace (notes on Babbage’s Analytical Engine)

Falhas Espetaculares de Software

- Ariane 5:
  - Explosão em seu primeiro voo
  - Causado pelo reuso de algumas partes de código de seu predecessor sem verificação adequada
- Therac-25:
  - Máquina de terapia de radiação
  - Devido a um erro de software, seis pessoas morreram de overdose
- Pentium FDIV:
  - Erro de projeto na unidade de divisão de ponto-flutuante
  - Intel foi forçada a oferecer substituição de todos os processadores defeituosos

Teste e Depuração

- Testar:
  - Avaliar o software pela observação de sua execução
- Falha de teste:
  - Execução de um teste que resulta em um defeito de software
- Depurar:
  - Processo de encontrar uma falha a partir da informação de um defeito

Objetivos do Teste de Software (níveis de maturidade)

- Nível 0: não existe diferença entre teste e depuração
- Nível 1: o objetivo do teste é demonstrar correção
- Nível 2: o objetivo do teste é demonstrar que o software não funciona
- Nível 3: o objetivo do teste não é provar algo específico, mas reduzir o risco da utilização do software
- Nível 4: teste é uma disciplina mental que ajuda todos os profissionais a desenvolver software de melhor qualidade

Nível 0

- Testar é o mesmo que depurar
- Não distingue entre comportamento incorreto e erros de programação
- Não ajuda a desenvolver software que é confiável ou seguro

Nível 1

- Propósito é demonstrar correção
- Correção é impossível de ser atingida
- O que se sabe no caso de não apresentar defeitos?
  - O software é bom ou os testes são ruins?

Nível 2

- Propósito é mostrar defeitos
- Procurar defeitos é uma atividade “negativa”
- Usualmente coloca desenvolvedores e testadores em relacionamentos conflituosos

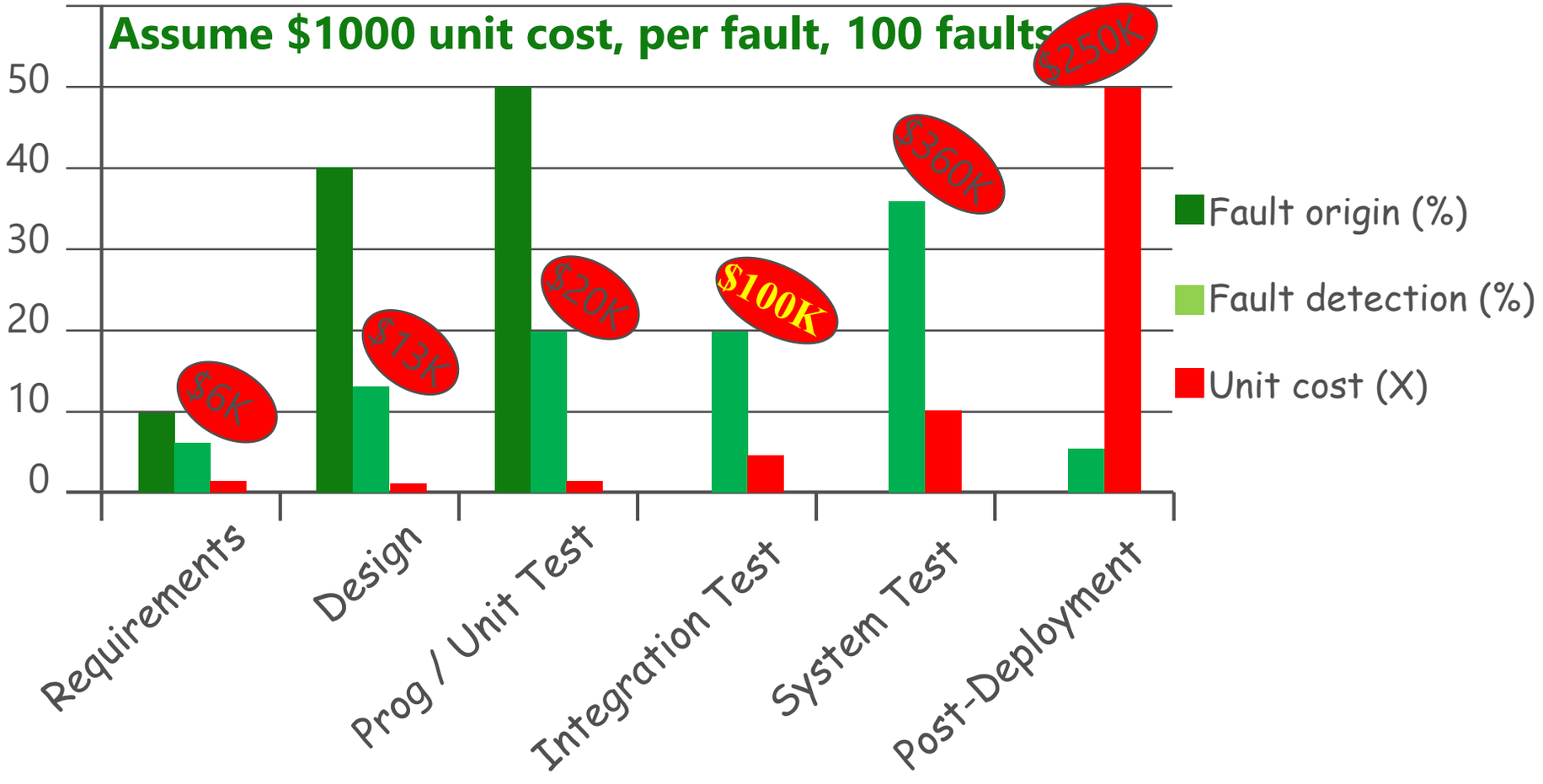
Nível 3

- Teste somente pode apresentar a presença de defeitos
- Sempre que utilizamos algum software, incorremos em algum risco
  - Risco pode ser pequeno e as consequências sem importância
  - Risco pode ser grande e as consequências catastróficas
- Desenvolvedores e testadores cooperam para reduzir o risco

Nível 4

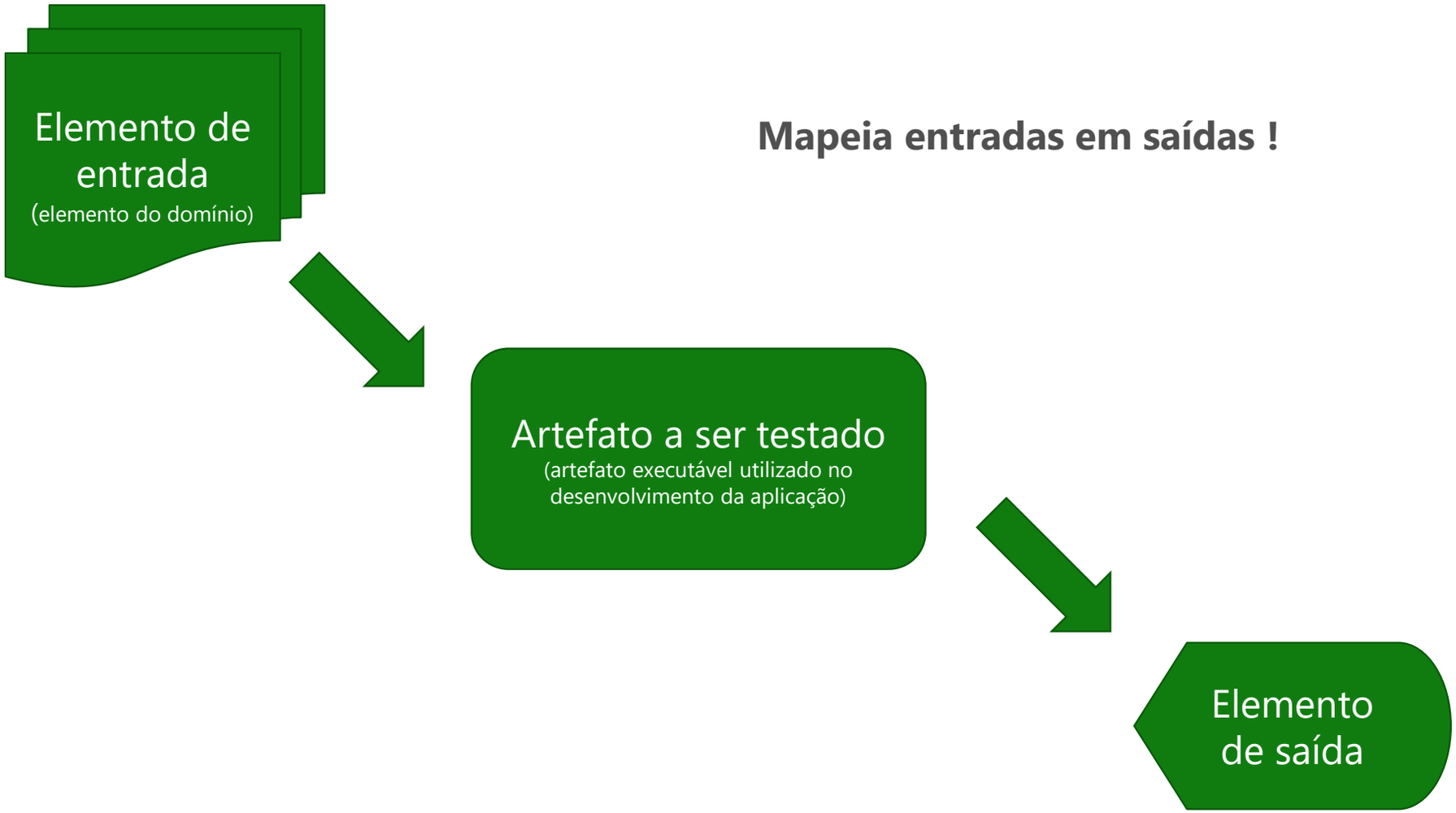
- Teste é somente um dos muitos meios para aumentar a qualidade do software
- Engenheiros de teste possuem grande responsabilidade e liderança técnica
  - Principal responsabilidade é medir e melhorar a qualidade do software produzido

Custo de Teste de Software



Software Engineering Institute; Carnegie Mellon University; Handbook CMU/SEI-96-HB-002

Artefatos de Software Como Uma Função



Mapeia entradas em saídas !

Elementos do Teste de Software

- O artefato
- A descrição do comportamento esperado
  - Normalmente proveniente de um oráculo
  - Oráculo é um mecanismo usado por testadores de software para determinar se um teste deve falhar ou não
- Observação da execução do artefato em teste
- Descrição do domínio funcional
  - Domínio de entrada: quais os possíveis valores de entrada para o artefato em questão
- Método para determinar se o comportamento observado corresponde ao comportamento esperado

Casos de teste

- Um caso de teste é um par ordenado: (d,S(d))
  - d é um elemento de um domínio D (d ∈ D)
  - S(d) é o resultado esperado para uma dada função de especificação quando d é utilizado como entrada.
- Normalmente pode ser representado sob forma matricial:

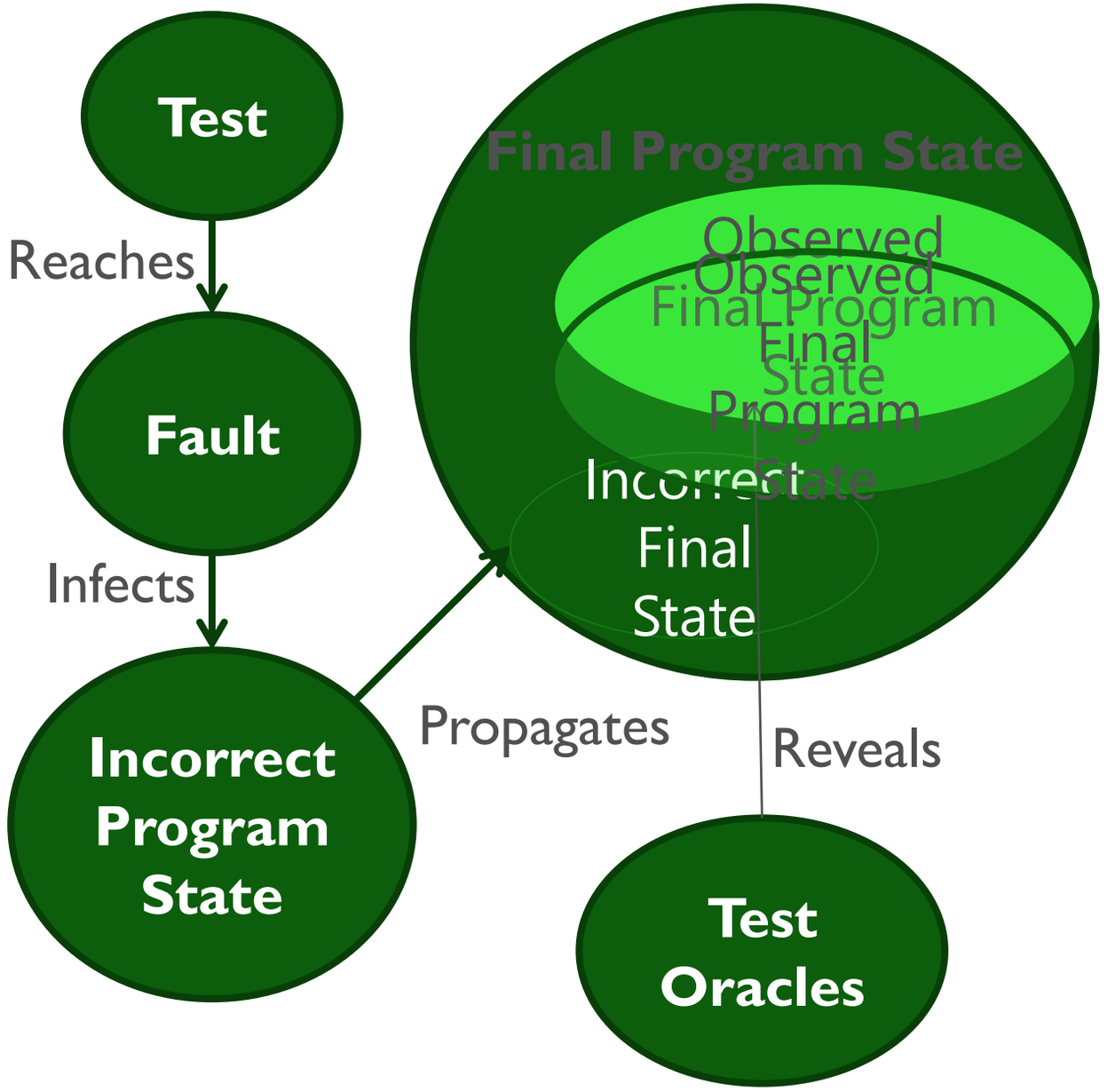
Valores de Entrada	Resultados Esperados
10032 25 "Jorge Andrade"	"Usuario cadastrado"
10033 0 "Luis Santos"	"Idade inválida – usuário não cadastrado"

Critérios de teste

- A verificação completa de um programa P envolve o teste de P com um conjunto de casos de teste T que compreende todos os elementos do domínio de entrada D.
- Como normalmente D é infinito ou muito grande utiliza-se um conjunto de critérios de teste que auxiliam o testador a definir casos de teste potencialmente reveladores de defeitos.

Modelo RIPR

- Reachability
- Infection
- Propagation
- Revealability



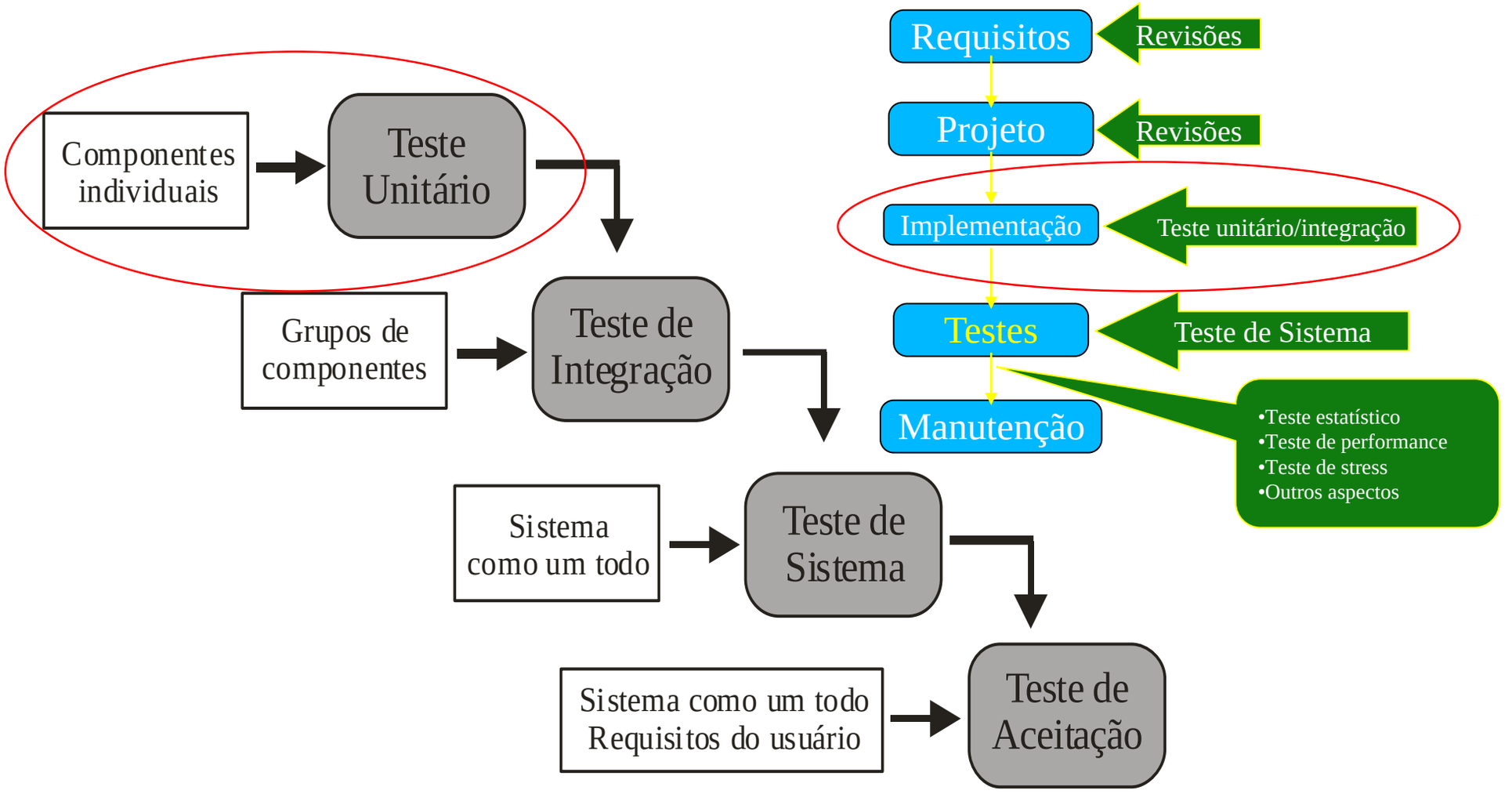
Atividades de Teste

- Atividades de teste de software:
  - Planejamento
  - Projeto dos casos de teste
  - Execução dos casos de teste
  - Avaliação dos resultados

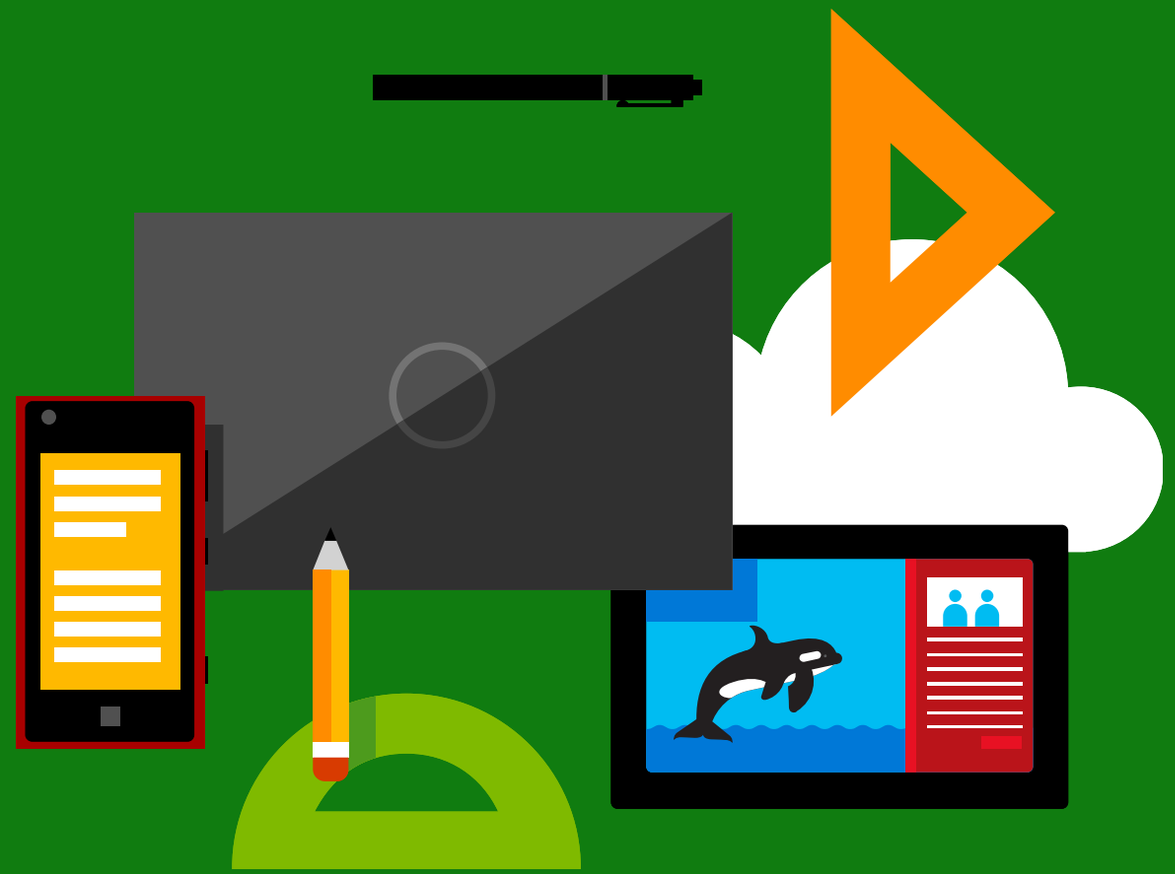
Fases de Teste

- As atividades de teste podem ser realizadas em cada uma das seguintes fases:
  - Teste de unidade ou unitário
    - Busca identificar defeitos na lógica e implementação de cada unidade de software isoladamente.
    - No caso de P.O.O. as unidades correspondem a classes.
    - Usam-se *drivers* e dublês.
  - Teste de integração
    - Visa descobrir defeitos nas interfaces das unidades durante a integração da estrutura do programa.
  - Teste de sistema
    - Procura identificar defeitos nas funcionalidades do sistema como um todo.

Teste Unitário no Ciclo de Vida de Desenvolvimento de Software



Teste Unitário



Entradas Para o Teste Unitário

- Especificação do módulo antes da sua implementação:
  - Fornece subsídios para o desenvolvimento de casos de teste.
  - Fundamental como oráculo.

- Código fonte do módulo:
  - Desenvolvimento de casos de teste complementares após a implementação do módulo.
  - Não pode ser usado como oráculo.

Artefatos Gerados pelo Teste Unitário

- Classes “drivers”
  - São as classes que contêm os casos de teste.
  - Procuram exercitar os métodos da classe “alvo” buscando detectar falhas.
  - Normalmente: uma classe “driver” para cada classe do sistema.
- Dublês (“mockups”)
  - Simulam o comportamento de classes necessárias ao funcionamento da classe “alvo” e que ainda não foram desenvolvidas.
  - Quando a classe correspondente ao “dublê” estiver pronta, será necessário re-executar o “driver” que foi executado usando-se o “dublê”.

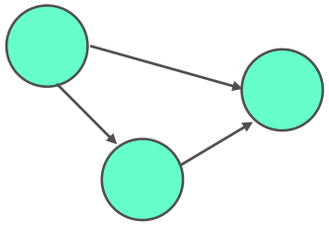
Critérios de Cobertura

- Mesmo pequenos programas possuem um número muito grande de entradas para que um teste seja exaustivo
- Exemplo: double computeMedia(int a, int b, int c)
  - Em uma máquina de 32-bits, cada variável possui 4 bilhões de valores possíveis
  - Acima de 80 octilhões (10^{27}) de testes possíveis
  - Matematicamente o espaço de valores de entrada é infinito
- Testadores procuram o menor conjunto de entrada que permite encontrar o maior número de problemas
- Critérios de cobertura fornecem um meio prático e estruturado de construir um espaço de entradas

Critérios de Cobertura

- Pesquisadores da área de teste apresentam dezenas de critérios, mas é possível cataloga-los em quatro grandes tipos:
  - Domínio de entrada (conjuntos)
  - Grafos
  - Expressões lógicas
  - Estruturas sintáticas (gramáticas)

```
A: {0, 1, >1}
B: {600, 700, 800}
C: {swe, cs, isa, infs}
```



```
(not X or not Y) and A and B
```

```
if (x > y)
    z = x - y;
else
   z = 2 * x;
```

Critérios de Cobertura

- Características de um bom critério de cobertura:
  - Deve ser razoavelmente simples de computar requisitos de teste de forma automática
  - Deve ser possível de gerar casos de teste de forma eficiente
  - O resultado dos testes devem revelar o maior número de falhas possível

Domínio de Entrada

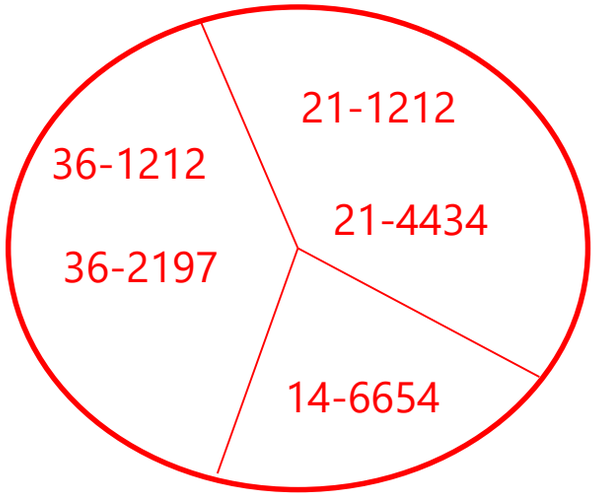
- Usam como entrada a especificação do módulo
- Especificação/Contratos:
  - É fundamental gerar casos de teste com valores válidos buscando verificar se o módulo se comporta como especificado
  - No caso de programação por contratos deve-se gerar casos de teste que busquem verificar se a implementação atende as especificações do contrato
- Não deve utilizar o código-fonte de implementação
  - Critérios baseados em grafos e lógica utilizam o código

Domínio de Entrada

- Valor Limite
  - Funciona bem quando o programa a ser testado é função de várias variáveis independentes que representam conjuntos que tenham uma relação de ordem.
  - Geração de casos de teste:
    - Fixa-se o valor de todas as variáveis menos uma em seus valores nominais
    - A variável escolhida assume os valores: {MIN,MIN+1,NOMINAL,MAX-1,MAX}
    - Exemplo:
    - v1:int $\in$ [10,20]
    - Casos de teste: {10, 11, 15, 19, 20}
  - Se existir mais de uma variável, deve-se testar as combinações possíveis entre elas
  - Se a variável for uma enumeração deve-se testar todos os valores

Domínio de Entrada

- Classes de equivalência
  - Princípio: dividir o domínio de entrada em subconjuntos de maneira que o comportamento de um dos membros do conjunto seja representativo do comportamento de todos os membros do conjunto.
  - Gerar pelo menos um caso de teste para cada classe
  - Ex: Nros de telefone (não tem relação de ordem) onde o prefixo indica a operadora.



Grafos

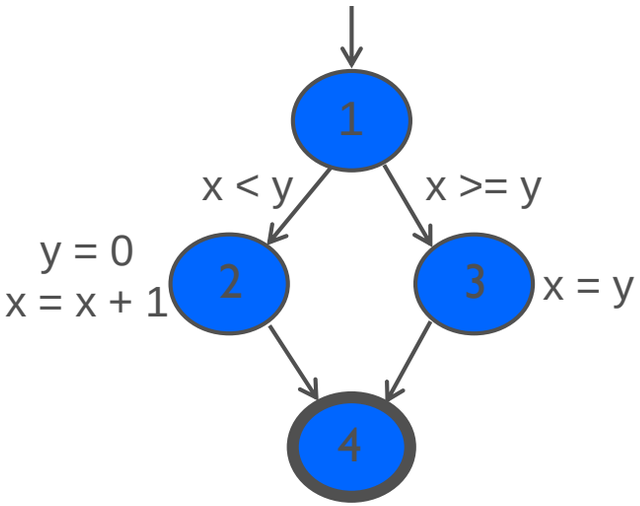
- São uma das estruturas mais comuns para estruturar testes
- Podem ser obtidos de diferentes fontes:
  - Grafos de fluxo de controle
  - Modelos UML
  - Máquinas de estado
  - Casos de uso
- Gerar casos de teste que garantam cobertura de nós e arcos

Grafos

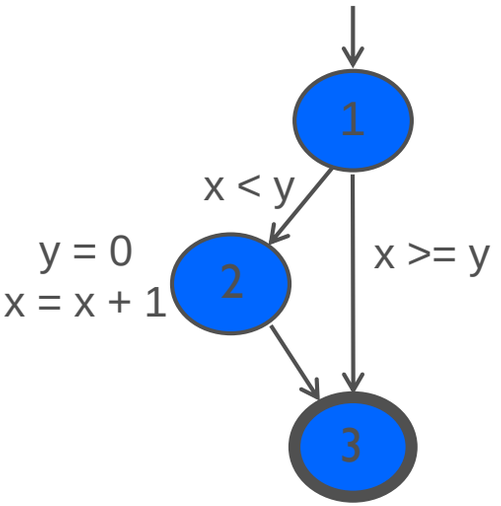
- Ao usar como entrada o código fonte:
  - Usado para refinar os casos de teste
  - Gerar o grafo de programa
  - Procurar garantir a cobertura do grafo de programa
    - Verificar se passa por todos os comandos
    - Verificar se exercita cada uma das opções de cada condição
    - Verificar se cada laço itera pelo menos k vezes

Grafos

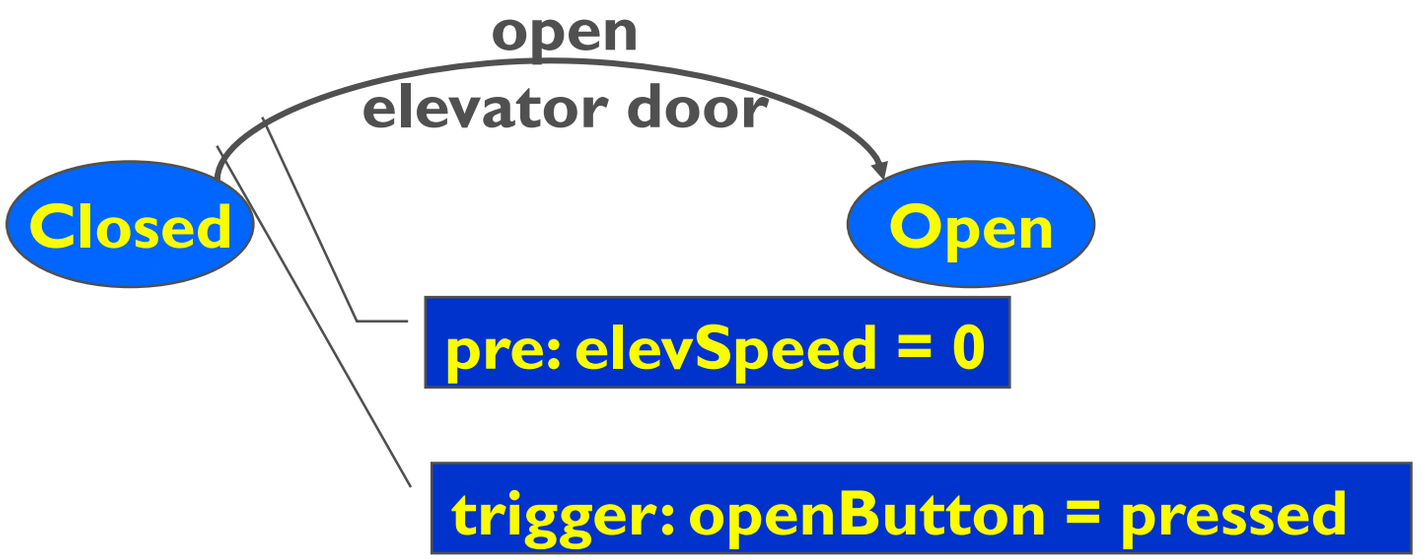
```
if (x < y)
{
   y = 0;
   x = x + 1;
}
else
{
   x = y;
}
```



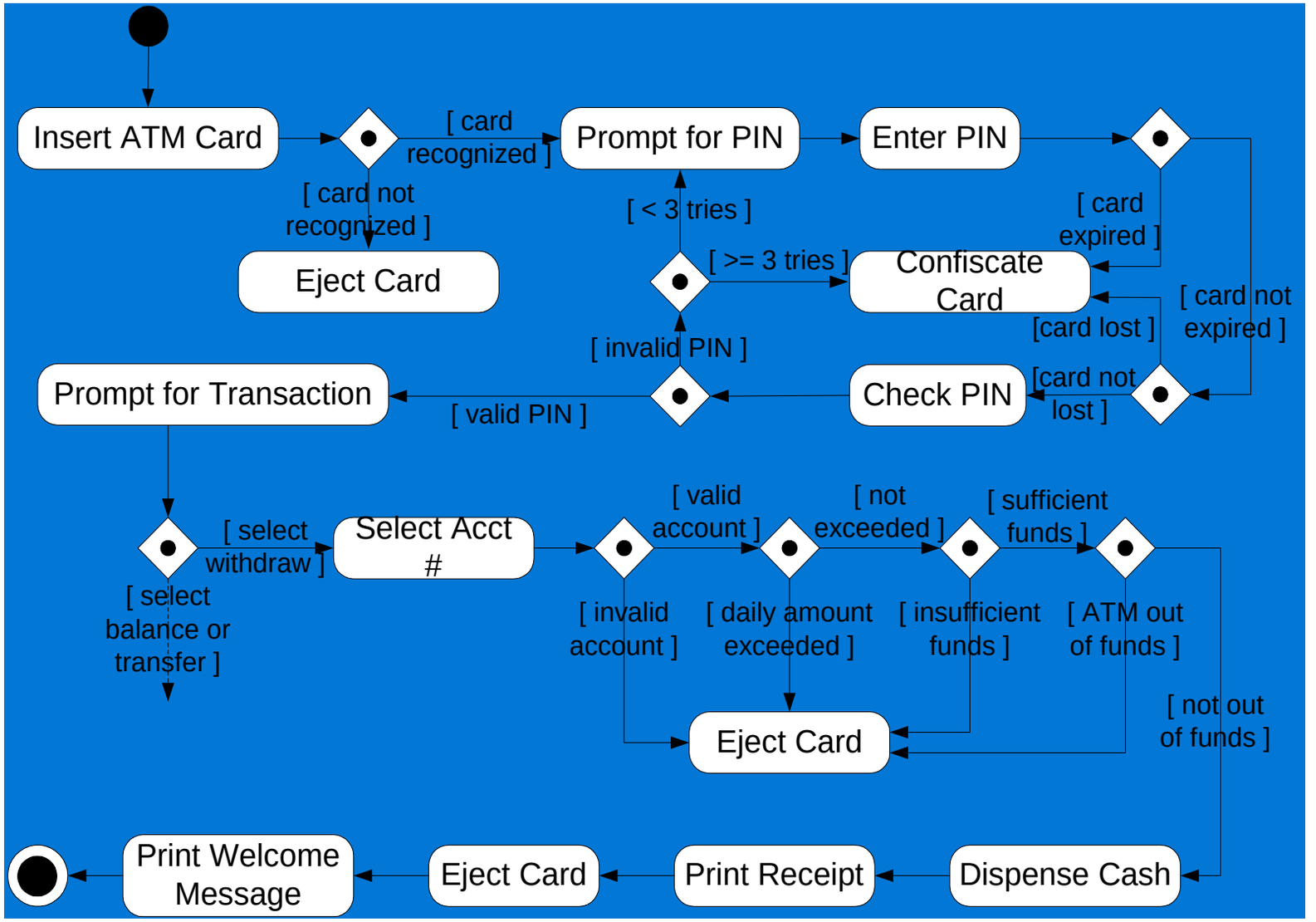

```
if (x < y)
{
   y = 0;
   x = x + 1;
}
```



Grafos


open
elevator door
Closed
Open
pre: elevSpeed = 0
trigger: openButton = pressed

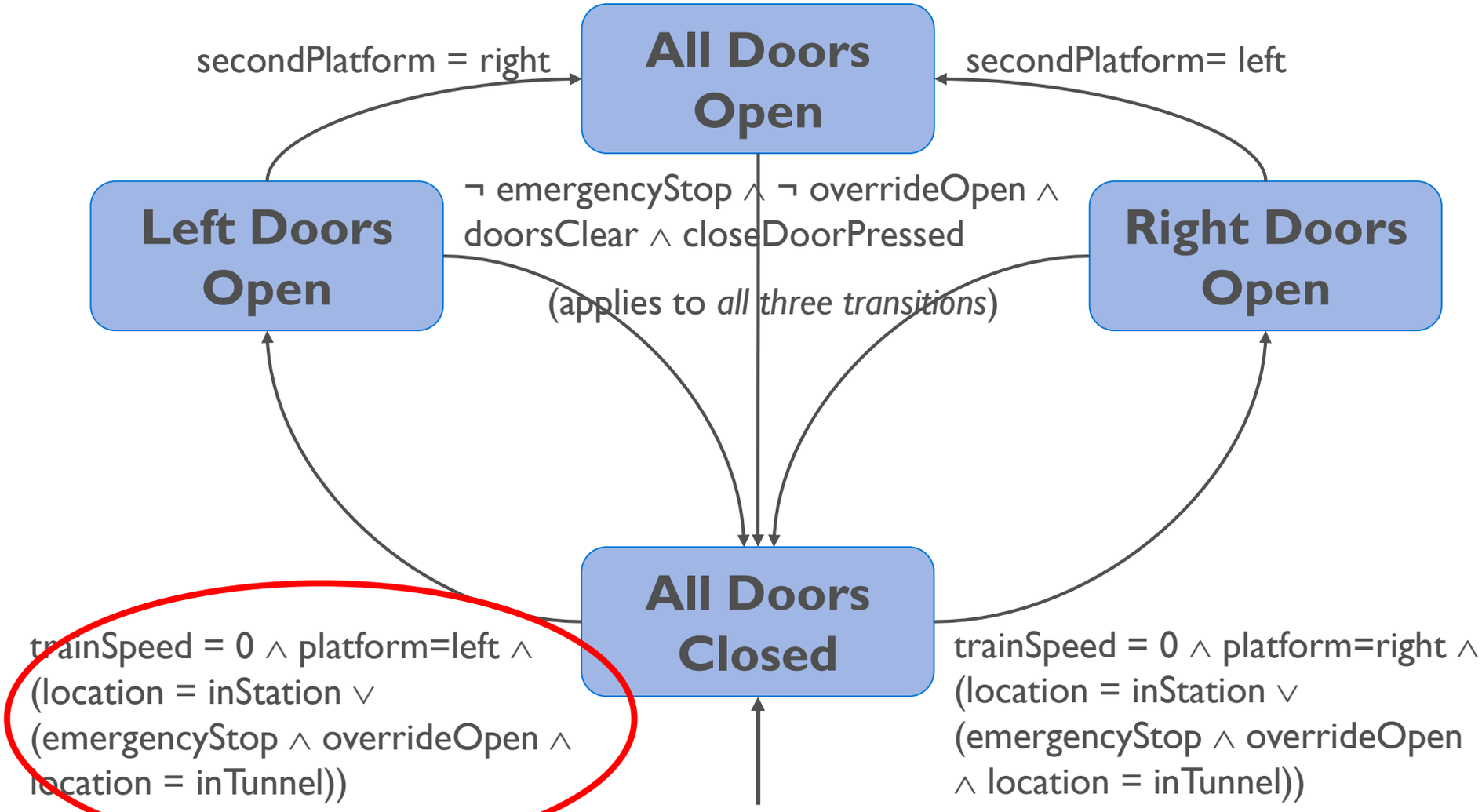
Grafos


Insert ATM Card
[card recognized]
[card not recognized]
Eject Card
Prompt for PIN
[< 3 tries]
[>= 3 tries]
Confiscate Card
Enter PIN
[card expired]
[card not expired]
[card lost]
[invalid PIN]
Prompt for Transaction
[valid PIN]
Check PIN
[card not lost]
[select withdraw]
[select balance or transfer]
Select Acct #
[valid account]
[not exceeded]
[sufficient funds]
[invalid account]
[daily amount exceeded]
[insufficient funds]
[ATM out of funds]
[not out of funds]
Eject Card
Print Welcome Message
Eject Card
Print Receipt
Dispense Cash

Expressões Lógicas

- Podem ser obtidas de diferentes fontes:
  - Comandos de decisão em programas
  - Máquinas de estado
  - Requisitos
  - Contratos

Expressões Lógicas



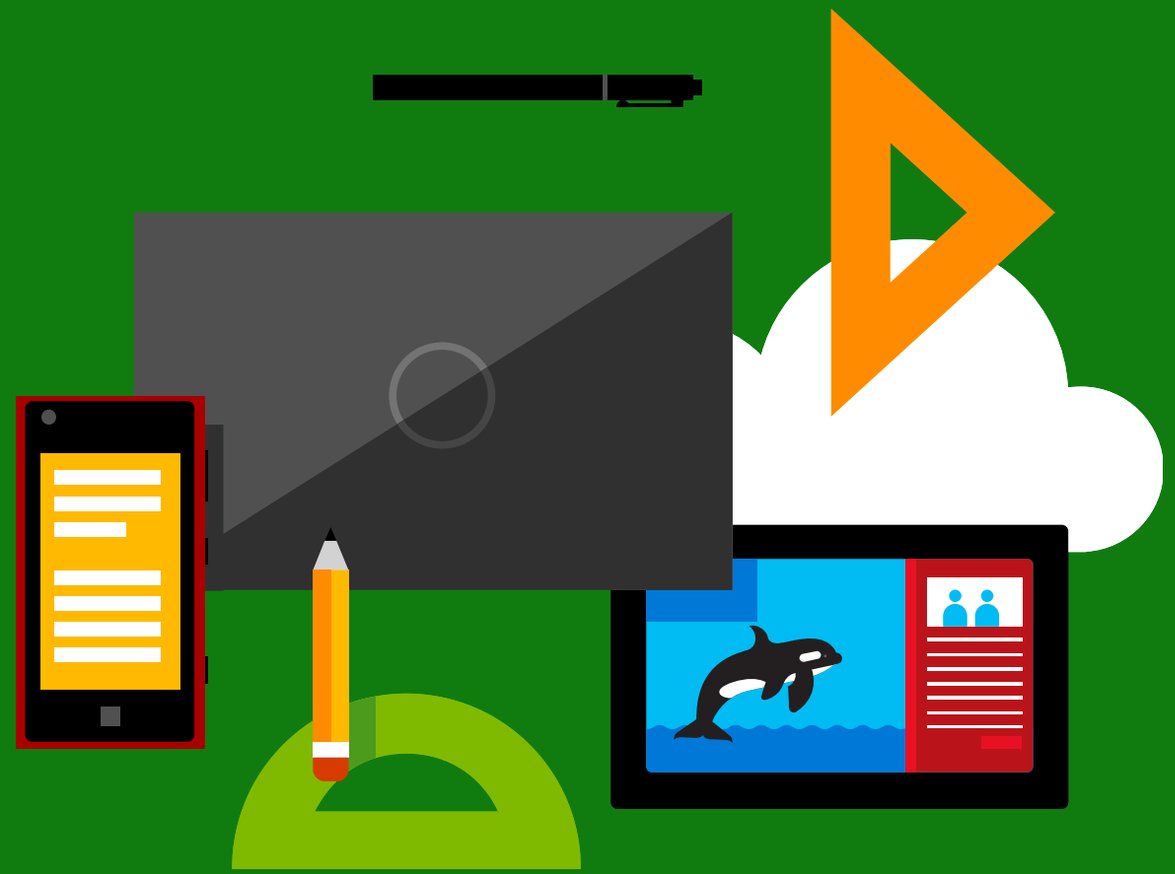
Estruturas Sintáticas

- Sintaxe é usualmente representada via gramáticas
  - Expressões regulares também são usuais
- Descrições sintáticas são usualmente obtidas a partir:
  - Programas
  - Elementos de integração
  - Documentos de projeto
  - Descrições de formatos de entradas

Critérios Adicionais

- Condições de erro
  - Gerar casos de teste que procurem gerar as condições de erro previstas (ex: geração de exceções)
- Valores inválidos
  - Gerar caso de teste com valores de entrada inválidos

JEST



JEST

Framework para realização de testes em JavaScript
Suporta testes unitários, dublês, cobertura de código, etc
Disponível em https://jestjs.io/