Metodologias de Teste de *Software*

Fernando Magno Quintão Pereira

10 de Novembro de 2010

A espiral de testes

Muitos processos de teste de *software* seguem a mesma espiral de desenvolvimento que vemos em *Engenharia de Software*, mas em ordem inversa.

1. Testes de unidade $\equiv$ Programação/codificação.
2. Testes de integração $\equiv$ Projeto.
3. Testes de validação $\equiv$ Levantamento de requisitos.

Questão

- Quando podemos começar a testar o *software*?
- Quando podemos parar de testá-lo?
- Quem é responsável por testar o *software*?
- Quais as vantagens e desvantagens de uma companhia contratar um grupo de testes independente?

Aprenda a "ler" os testes

Um teste de *software* fornece muito mais informação que simplesmente: "o programa funciona/não funciona".
Que informações vale a pena guardar dos resultados de um teste de *software*?

Aprenda a "ler" os testes

Um teste de *software* fornece muito mais informação que simplesmente: "o programa funciona/não funciona".
Que informações vale a pena guardar dos resultados de um teste de *software*?

1. Tempo médio para encontrar uma falha.
2. Tempo médio para reparar um defeito.
3. Variação na frequência de erros encontrados.
4. Número de horas gastas no preparo de testes.

Que outras informações podemos inferir a partir de testes?

Testes de unidade

Testes de unidade testam as menores unidades compiláveis que compõem um sistema. Diferentes aspectos de um programa podem ser testados:

- Sanitização de dados de entrada.
- Condições de contorno.
- Tratamento de exceções.

Questão

Cite exemplos de erros que são encontrados por testes de unidade.

- Comparação entre tipos de dados diferentes.
- ...

Questão

Cite exemplos de erros que são encontrados por testes de unidade.

- Comparação entre tipos de dados diferentes.
- ...
- Uso incorreto da precedência entre operadores lógicos.
- Expectativa de igualdade quando precisão pode tornar igualdade improvável.
- Comparação incorreta de variáveis.
- Terminação inexistente de *loop*.
- Modificação incorreta de variável de indução no interior do *loop*.

Questão

Testes de unidade em geral são realizados sobre um módulo do programa, quando o programa inteiro ainda não foi terminado.

- Como isto é possível?

Testes de Integração

Testes de integração são técnicas sistemáticas para construir a arquitetura de *software*, ao mesmo tempo em que testes são realizados para descobrir erros associados aos pontos de junção.

- Tudo bem, todos os componentes do sistema já foram testados individualmente. O que pode dar errado?
- Mas se colocarmos todos os módulos juntos, de uma vez só, e algo dar errado, como descobrir o erro?

Integração *Top Down*

- O módulo de controle principal é usado como um *driver* de testes.
- *Stubs* são substituídos um de cada vez.
- Novos testes são executados conforme cada novo componente é inserido.
- Findo um teste, um novo *stub* é substituído por um componente real.
- Testes de regressão são executados ao final, para garantir que novos erros não foram inseridos durante o processo.

Integração *Bottom Up*

- Components de baixo nível são combinados em grupos que realizam funções comuns.
- *Drivers* são escritos para coordenar os testes de cada grupo, regulando a entrada e a saída de dados.
- *Drivers* são removidos, enquanto grupos são combinados, formando-se grupos maiores.

Testes de Regressão

Testes de regressão são formados pela re-execução de testes que já foram executados, com o fim de garantir que a adição de novos componentes não foi responsável por novos *bugs*.

- Como estes testes deveriam ser organizados?
- Será que todos os testes antigos deveriam ser re-executados?
- Será que novos testes deveriam ser adicionados?
- Será que alguns testes deveriam ser enfatizados?

Testes de Validação

Testes de validação procuram descobrir falhas no *software* no contexto em que ele é usado. O foco destes testes são nos requisitos, ou seja, na parte do sistema com a qual o usuário final irá interagir.

- Em suas próprias palavras, quais são as diferenças entre validação e verificação?
- Como saber se um programa atende os requisitos que motivaram sua criação?
- Como testes de validação deveriam ser executados?

Validação Alfa e Validação Beta

A validação alfa é executada no próprio local de desenvolvimento do *software*. Os desenvolvedores acompanham os usuários, enquanto estes utilizam o programa desenvolvido.
A validação beta é conduzida no local onde o sistema será usado de fato. Normalmente os desenvolvedores não estão presentes nesta etapa.

- Testes de unidade, testes de integração, testes de validação, ... Existe algo mais que ainda precise ser testado?

Testes de Sistema

O *software* desenvolvido ainda precisa ser testado em face a situações excepcionais ou de tensão. Que situações são estas?

Testes de Sistema

O *software* desenvolvido ainda precisa ser testado em face a situações excepcionais ou de tensão. Que situações são estas?

- Recuperação após falhas.
- Segurança: por exemplo, tentativa de acesso não autorizado.
- Desempenho.
- Falta de recursos: exemplos?

Testes de Sistema

O *software* desenvolvido ainda precisa ser testado em face a situações excepcionais ou de tensão. Que situações são estas?

- Recuperação após falhas.
- Segurança: por exemplo, tentativa de acesso não autorizado.
- Desempenho.
- Falta de recursos: exemplos?
  - Banda de rede muito pequena
  - Velocidade de *clock* insuficiente
  - Pouca memória disponível

Depuração

O teste de *software* pode levar à descoberta de *bugs*, os quais precisam ser depurados. Porque a depuração é uma arte tão difícil?

Depuração

O teste de *software* pode levar à descoberta de *bugs*, os quais precisam ser depurados. Porque a depuração é uma arte tão difícil?

- Causas e sintomas podem estar localizados em partes distintas do código.
- Os sintomas podem desaparecer temporariamente enquanto outros erros são corrigidos.
- Os sintomas podem ser causados por falhas que não são necessariamente erros, como arredondamento, por exemplo.
- Os sintomas podem ser causados por erros humanos, que são difíceis de traçar.
- As entradas que causam os erros podem ser difíceis de reproduzir – o pesadelo do não determinismo.
- Os sintomas podem ser intermitentes.
- Os sintomas podem ser devido a uma conjunção de eventos que executam em processadores diferentes.

O guia do depurador eficiente

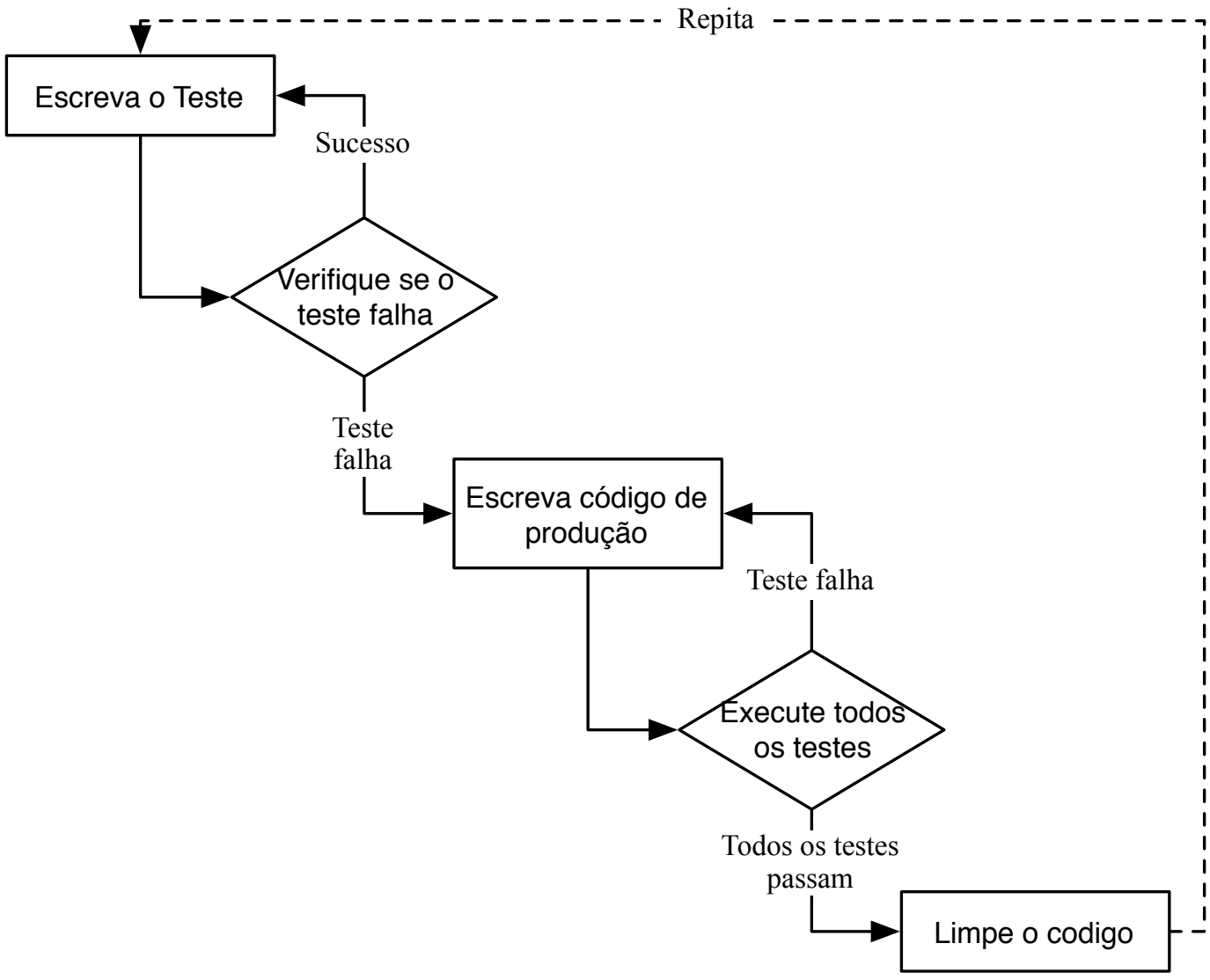
- Quais são boas práticas de depuração?
- Algum exemplo da vida real?

O guia do depurador eficiente

- Quais são boas práticas de depuração?
- Algum exemplo da vida real?

1. Força bruta.
2. *Backtracking*.
3. Eliminação de causas.

Desenvolvimento voltado à testes



Questão 1

Uma forma de produzir bons testes é encontrar um conjunto de entradas que exercitem todos os caminhos em um programa.

1. Quantos caminhos existem?
2. Quantas entradas devem ser criadas para o programa abaixo?

```
public class Base1 {
  public static void main(String[] args) {
    if (args.length == 0) {
      System.err.println("Syntax: java Base1 [pattern]");
      System.exit(1);
    } else {
      int sum = 0;
      for (String s : args) {
        for (short c = 0; c < s.length(); c++) {
          sum += s.charAt(c);
        }
      }
      System.out.println("Sum = " + sum);
    }
  }
}
```

Questão 2

Implemente o tipo abstrato de dados `Graph`. Comece pelos testes:

1. Que operações devem ser testadas?

Utilize uma metodologia de desenvolvimento voltada para os testes. `junit` pode vir acalhar aqui:

- `javac -cp .:junit.jar TestGraph.java`
- `java -cp .:junit.jar TestGraph`

Questão 3

Como você implementaria cada um destes testes?

1. Adicionar um nodo ao grafo.
2. Adicionar uma aresta ao grafo.