

DIM0436

Revisões e testes

20141023

Sumário

1 Revisões e inspeções

2 Introdução ao teste de software

1 Revisões e inspeções

2 Introdução ao teste de software

Citação

There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.

– C.A.R. Hoare (1980)

Controlar a qualidade do software

Prova formal

- Demonstração matemática rigorosa
- A especificação deve também ser formal

Argumentação

- Verificação baseada em princípios de prova
- A especificação deve ser (suficientemente) rigorosa e completa

Revisões

- Controle da qualidade do software
- Qualquer artefato pode ser revisado

O que pode ser revisado ?

- Especificações

O que pode ser revisado ?

- Especificações
- Projetos

O que pode ser revisado ?

- Especificações
- Projetos
- Código

O que pode ser revisado ?

- Especificações
- Projetos
- Código
- Casos de teste

O que pode ser revisado ?

- Especificações
- Projetos
- Código
- Casos de teste
- Documentação

O que pode ser revisado ?

- Especificações
- Projetos
- Código
- Casos de teste
- Documentação
- Planos de desenvolvimento

O que pode ser revisado ?

- Especificações
- Projetos
- Código
- Casos de teste
- Documentação
- Planos de desenvolvimento
- Processo de desenvolvimento

O que pode ser revisado ?

- Especificações
- Projetos
- Código
- Casos de teste
- Documentação
- Planos de desenvolvimento
- Processo de desenvolvimento
- Padrões

Tipos de revisão

Pelo próprio autor

- ✗ Muitas faltas passarão despercebidas
- ✗ Erros de interpretação não serão percebidas

Por colegas

- ✗ Controle superficial para não gera atrito com o colega
- ✗ Falta de experiência para revisar o código

Progressivas

- Realizadas por um grupo de revisores com habilidades diferentes
- ✓ Pontos de vista diferentes reduzem as faltas despercebidas

Walkthrough

- O autor narra o comportamento do artefato e os demais participantes criticam, comentam, apontam riscos, sugerem melhorias ...
- ✗ Todos os participantes devem ter uma cópia antes da reunião

Revisões: checklist

- Existe algum elemento cuja especificação não deixe clara a sua intenção ?
- Existe algum elemento com intenção ambígua ?
- Existe redundância de especificação ? (**DRY!**)
- Existe elementos cuja abrangência de intenção possa ser reduzida sem comprometer o seu papel ?
- Existem elementos com itens de especificação não relacionados com a sua intenção ?

Inspeções

Definição

Uma inspeção é uma revisão realizada de acordo com um procedimento definido e documentado

Atores da inspeção

- Moderador
- Desenvolvedor
- Inspetor
- Controlador da qualidade

Revisão de inspeção \neq depuração

Numa reunião de inspeção:

- Evitar discussões sobre detalhes técnicos

Revisão de inspeção \neq depuração

Numa reunião de inspeção:

- Evitar discussões sobre detalhes técnicos
- Maximizar a produtividade da reunião de inspeção

Revisão de inspeção \neq depuração

Numa reunião de inspeção:

- Evitar discussões sobre detalhes técnicos
- Maximizar a produtividade da reunião de inspeção
- Impedir a introdução de faltas em virtude de decisões tomadas sem cuidado.

Revisão de inspeção \neq depuração

Numa reunião de inspeção:

- Evitar discussões sobre detalhes técnicos
- Maximizar a produtividade da reunião de inspeção
- Impedir a introdução de faltas em virtude de decisões tomadas sem cuidado.
- Assegurar que o resultado corrigido esteja dentro dos padrões de qualidade esperados.

1 Revisões e inspeções

2 Introdução ao teste de software

Técnicas de análise

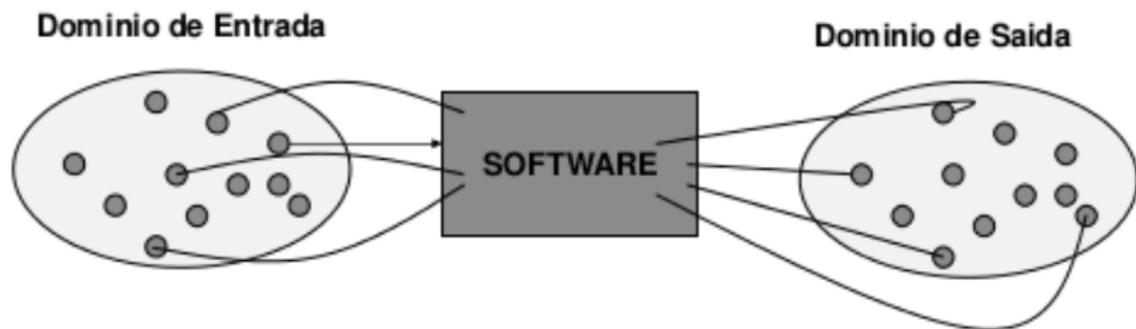
Neutralizar o teorema de Rice

- **Testes** (esta aula + as 3 seguintes)
 - ▶ Muito usados na indústria até hoje. Não garante a correção do código mas acha contraexemplos
 - ▶ *Program testing can be used to show the presence of bugs, but never to show their absence!*
– Dijkstra, 1969
- **Model-checking**
 - ▶ Analisar não o programa mesmo mas modelos dele (por exemplo, a política de segurança)
 - ▶ Perda de precisão possível
- **Métodos dedutivos**
 - ▶ Raciocinar precisamente, por dedução, sobre o programa, e extrair os problemas "difíceis" (condições de verificação)
 - ▶ Geralmente não automático
- **Interpretação abstrata**
 - ▶ Realizar aproximações da semântica concreta
 - ▶ Perda de precisão possível

Testar em perguntas

- O que é testar ?
- Por que testar ?
- Quando testar ?
- O que testar ?
- Como testar

Testes



Primeiras definições

O objetivo dos testes é mostrar que erros não estão presentes no software.

Testes procuram assegurar que o programa faz o que se supõe que ele faça.

Impossibilidades

Multiplicação

```
long long mult(long long, long long)
```

Testar exaustivamente

- O intervalo de variação para cada parâmetro é
 $I = [-9223372036854775807, 9223372036854775807]$
- Tem $I \times I$ pares a serem testados, i.e.
 $340282366920938463426481119284349108225 > 10^{38}$ entradas ...
- Um dia $\approx 10^5$ s i.e precisamos de 10^{33} dias para testar exaustivamente mult
...
- O sol ainda tem uma expectativa de vida de 10^8 anos ...

Então, o programa tem falhas

Definição

Objetivo

Num teste, o programa executa-se com o objetivo de encontrar **defeitos**

Hipótese de trabalho

Todo programa tem erros.

Impacto da definição sobre o testador

- Se o testador pensar que a sua tarefa é verificar a ausência de erros

Impacto da definição sobre o testador

- Se o testador pensar que a sua tarefa é verificar a ausência de erros
 - ▶ Vai trabalhar para isso

Impacto da definição sobre o testador

- Se o testador pensar que a sua tarefa é verificar a ausência de erros
 - ▶ Vai trabalhar para isso
 - ▶ Vai deixar escapar erros que existem (inconscientemente)

Impacto da definição sobre o testador

- Se o testador pensar que a sua tarefa é verificar a ausência de erros
 - ▶ Vai trabalhar para isso
 - ▶ Vai deixar escapar erros que existem (inconscientemente)
- Se a tarefa é encontrar problemas

Impacto da definição sobre o testador

- Se o testador pensar que a sua tarefa é verificar a ausência de erros
 - ▶ Vai trabalhar para isso
 - ▶ Vai deixar escapar erros que existem (inconscientemente)
- Se a tarefa é encontrar problemas
 - ▶ Vai trabalhar para isso

Impacto da definição sobre o testador

- Se o testador pensar que a sua tarefa é verificar a ausência de erros
 - ▶ Vai trabalhar para isso
 - ▶ Vai deixar escapar erros que existem (inconscientemente)
- Se a tarefa é encontrar problemas
 - ▶ Vai trabalhar para isso
 - ▶ Continuará o trabalhos para encontrar mais!

Chaos Monkey (Netflix)

Funcionamento

- Netflix construiu o sistema Chaos Monkey para apagar aleatoriamente serviços na sua própria infraestrutura
- Feito durante os horários de trabalho, nos dias úteis
- Assim, erros/fraquezas são identificados e analisados à vontade e não no meio da noite quando eles acontecem de verdade.

Em resumo

the best defense against major unexpected failures is to fail often.
– Netflix blog

Dropbox

Funcionamento

- Os times executam regularmente os sistemas com estresse adicional (e artificial)
- Se um limite do sistema for atingida e causo erros, o problema pode ser resolvido à vontade, apagando o estresse adicional.

Em resumo

This created a much less stressful situation than having to firefight against the same issues caused by production traffic that they couldn't just turn off.

Por que testar ?

Citação

Beware of bugs in the above code; I have only proved it correct, not tried it.

– Donald Knuth

Dados estatísticos

- Beizer estimou que testes tomam entre 30% e 90% do **custo** de desenvolvimento.
- Santhanan and Hailpern relata que de 50% a 75% do custo de desenvolvimento envolve teste e depuração
- Microsoft: 1 desenvolvedor $\rightarrow \approx$ 1 testador

With a ratio of testers:developers of 1:1, those testers keep busy by testing the product (...) - and that involves automation, a lot of automation.

Quando testar ?

Tip 62

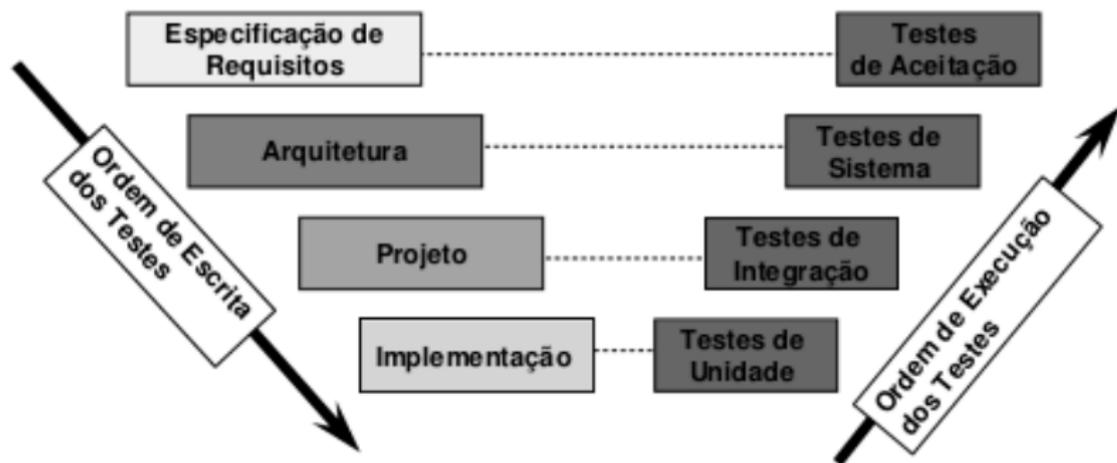
Test Early. Test Often. Test Automatically.

Tests that run with every build are much more effective than test plans that sit on a shelf.

– The Pragmatic Programmer

Então ?

- Testes eram uma etapa posterior a implementação : **big bang testing**
- Hoje, testes são um processo paralelo ao processo de desenvolvimento.



Tip 63

Coding Ain't Done 'Til All the Tests Run
– *The Pragmatic Programmer*

Como testar

Tip 66

Find Bugs Once

Once a human tester finds a bug, it should be the last time a human tester finds that bug. Automatic tests should check for it from then on.

– The Pragmatic Programmer

Categorias de testes

Tipos

- Teste de unidade
- Teste de integração
- Teste de desempenho
- Teste de usabilidade
- ...

Estrategias

- Testes de caixa preta
- Testes de caixa branca
- Testes de caixa cinza

Teste de unidade

Definição

- Um **teste de unidade** é um código de teste ao nível de um módulo (função, objeto, ...).
- O objetivo é testar componentes individuais
- São testados: interface, condições no limite, tratamento dos erros dentro da unidade

Teste de unidade contra contratos

Exemplo

```
/*@ requires x >= 0;  
double sqrt(double x);
```

Teste de unidade contra contratos

Exemplo

```
//@ requires x >= 0;  
double sqrt(double x);
```

O que testar ?

- Um x negativo, para testar a rejeição dele
- Um $x = 0$, para verificar que ele seja aceitado (valor limit)
- Vários valores entre 0 e o maior valor possível para verificar que a diferença é menor do que uma fração de x .

Teste de integração

- O objetivo dos **testes de integração** é mostrar que os subsistemas que compõem o projeto funcionam e funcionam bem um com o outro.
- A integração e os testes são feitos de forma incremental
- São testados: a interface entre os módulos
- A presença de contratos já testados (por testes unitários) facilita os testes de integração
- Muito semelhante aos testes de unidade, salvo que eles operam num subsistema inteiro.

Abordagens

Top-down

- ✓ localização dos erros simples
- ✓ não precisa do programa inteiro
- ✗ módulos críticos (em cima da hierarquia) são testados no fim do desenvolvimento
- ✗ impossibilidade de prototipar cedo

Bottom-up

- ✓ localização dos erros simples
- ✓ não precisa do programa inteiro
- ✓ módulos críticos são testados com uma maior prioridade
- ✗ muitos *stubs*
- ✗ módulos em baixo da hierarquia são menos testados

Outros testes

Teste funcional

- Testar a funcionalidade geral (contratos!)
- Condições válidas e inválidas

Teste de desempenho

- Verificar o tempo de reposta e de processamento para diferentes configurações
- Geralmente feito paralelamente ao **teste de estresse**

Teste de recuperação de falhas

- O software é forçado a falhar de diversas maneiras (Chaos Monkey!).
- Verificar a recuperação de falhas

Teste de usabilidade

- Verifica a interface com o usuário
- Navegação, consistência, aderência a padrões

Como testar ?

- O testador trabalha para encontrar **defeitos**
- É impossível testar todas as entradas possíveis
- Para aumentar as chances de encontrar uma falha, tem basicamente 2 estratégias
 - ▶ Testes de caixa preta
 - ▶ Testes de caixa branca

Cores de caixas

Testes de caixa preta

- Testes de caixa preta = testes funcionais
- Testes *Data-driven* ou *IO-driven*
- Os casos de testes manipulam apenas entradas e saídas

Testes de caixa branca

- Desenvolvedor precisa conhecer o funcionamento interno do programa
- Teste os caminhos lógicos do programa (critérios de cobertura)
- Fluxograma!

Testes de caixa cinza

- Conhecimento parcial do funcionamento interno
- Combinação das abordagens de caixa preta com caixa branca quando o funcionamento interno é conhecido
- Uso duma linguagem de especificação
- Adequado para testar aplicações web ou teste funcional.

Testes de caixa branca

Tip 65

Test State Coverage, Not Code Coverage

Identify and test significant program states. Just testing lines of code isn't enough.

– The Pragmatic Programmer

```
int test(int a, int b) {  
    return a / (a + b);  
}
```

- Tem só um caso de teste que gera um defeito
- O conhecimento da cobertura de linha de código não é suficiente.
- Testar é **difícil**

Como testar ?

Testes de regressão

- Um teste de regressão compara as saídas correntes com as valores conhecidos.
- Assegura que os elementos que funcionavam ontem continuam funcionando

Dados de testes

- Dados reais: dados **típicos** do uso
- Dados artificias
 - ▶ Se precisar de muitos dados, são necessários
 - ▶ Para testar as condições nos limites
 - ▶ Se precisar de dados com propriedades estatísticas

Como testar ?

Testar sistemas gráficos

- Ferramentas dedicadas
- Scripts especiais para gerar/capturar eventos
- Usar testes manuais
- Código modular ajuda especialmente para poder testar a lógica da aplicação sem testar a parte gráfica e isolar os bugs somente gráficos

Testar meticulosamente

- Como saber se o código foi suficientemente testado ?

Como testar ?

Testar sistemas gráficos

- Ferramentas dedicadas
- Scripts especiais para gerar/capturar eventos
- Usar testes manuais
- Código modular ajuda especialmente para poder testar a lógica da aplicação sem testar a parte gráfica e isolar os bugs somente gráficos

Testar meticulosamente

- Como saber se o código foi suficientemente testado ?
- Não sabe e nunca vai saber

Como testar ?

Testar sistemas gráficos

- Ferramentas dedicadas
- Scripts especiais para gerar/capturar eventos
- Usar testes manuais
- Código modular ajuda especialmente para poder testar a lógica da aplicação sem testar a parte gráfica e isolar os bugs somente gráficos

Testar meticulosamente

- Como saber se o código foi suficientemente testado ?
- Não sabe e nunca vai saber
- Veja ??

Testar os testes

Tip 64

Use Saboteurs to Test Your Testing.

Introduce bugs on purpose in a separate copy of the source to verify that testing will catch them.

– The Pragmatic Programmer

Uma última dica

Tip 61

Don't Use Manual Procedures.

Humans are not reliable. A shell script or batch file will execute the same instructions, in the same order, time after time. – The Pragmatic Programmer

Resumo

1 Revisões e inspeções

2 Introdução ao teste de software

Referências

-  Andrew Hunt and David Thomas, *The pragmatic programmer: From journeyman to master*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
-  Glenford J. Myers and Corey Sandler, *The art of software testing*, John Wiley & Sons, 2004.

Perguntas ?



<http://dimap.ufrn.br/~richard/dim0436>