Verificação de Software e Teste

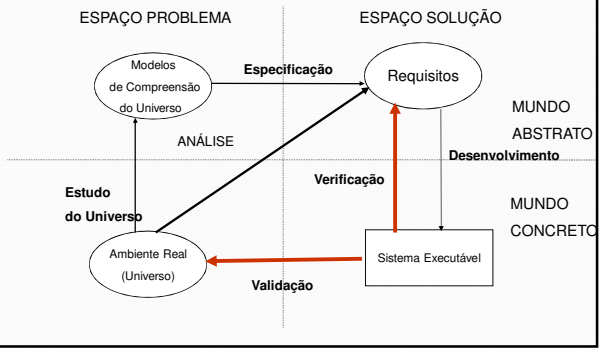
Karin Becker
Engenharia de Software N
Instituto de Informática - UFRGS

Inclui slides do Prof. Marcelo Pimenta, Erika Cota (UFRGS), adaptados com permissão

Verificação e Validação (V&V)

- Verificação [SEI-CMMI]
  - Confirmação que os produtos de trabalho refletem apropriadamente os requisitos especificados para eles
  - "você construiu corretamente o produto?"
- Validação [SEI-CMMI]
  - Confirmação que o produto fornecido (ou quando fornecido) irá atender seu uso pretendido
  - "você construiu o produto correto?"

Testes no Desenvolvimento de Software



Confiança de V&V

- Processo de V&V estabelece a confiança de que o software está pronto para seu propósito
- Depende do propósito do sistema, das expectativas dos usuários e do ambiente de mercado
  - Função de software
    - O nível de confiança depende de quão crítico é o software para uma organização
  - Expectativas do usuário
    - Os usuários podem ter baixas expectativas de certos tipos de software
  - Ambiente de mercado
    - Colocação de um produto para o mercado mais cedo pode ser mais importante que descobrir defeitos no programa

V&V

Static V&V:

- não envolve a execução do produto
- visa determinar propriedades do produto válidas para qualquer execução do produto final
- *This deals with Software Inspections.*
- Mechanisms used for this purpose:
  - *Program Inspections*
  - *Mathematical/Model Based Verification (MBV)*
  - *Static Program Analyzers (SPA)*
  - *Cleanroom Software Development*

Dynamic V&V:

- envolve a execução do produto (código ou modelo executável)
- visa encontrar falhas ou erros no produto
- This deals with Software Testing.
- Testing methods used for this purpose are:
  - Defect Testing
  - ValidationTesting

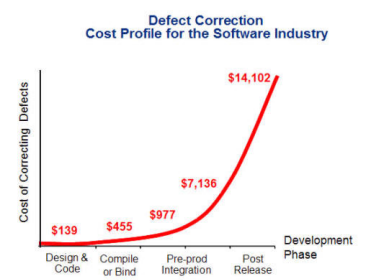
V&V: Fazer ou não fazer?

- Permite encontrar/evitar falhas
- Melhora a qualidade dos produtos
- Torna os requisitos mais estáveis
- Permite acompanhamento contínuo da qualidade e da produtividade
- Facilita o gerenciamento

MAS...

- Não sai de graça!
- Logo, reduz lucro ...

Custo do defeito na indústria



Source: Basili and Boehm, Software Defect Reduction Top 10 List, IEEE Computer Society, vol. 34, no. 1, Jan. 2001, pp. 135-137

V&V: Fazer ou não fazer?

- Baixa qualidade significa altos custos
- Encontrar e corrigir um defeito após a entrega do produto pode custar até 100 vezes mais do que na fase de requisitos
- Projetos atuais gastam de 40 a 50% de seus esforços em atividades de teste
  - Cerca de 80% do re-trabalho vem de 20% dos defeitos
  - Cerca de 80% dos defeitos vem de 20% dos módulos

8

Fontes de Erros

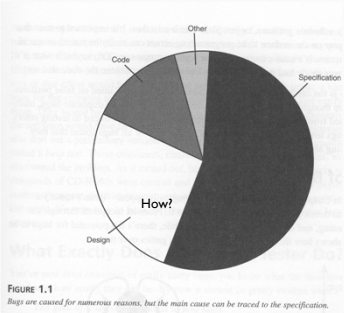


FIGURE 1.1
Bugs are caused for numerous reasons, but the main cause can be traced to the specification.

(source: "Software Testing", Ron Patton, 2006)

Teste

10

Teste

- Teste é a atividade de executar um software com o objetivo de **revelar** falhas
  - Falha: desvio do comportamento especificado
  - Erro: origem da falha
- Teste **não prova** que o programa está correto
- Teste **não conserta** o erro, apenas descreve a falha

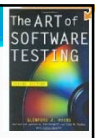
11

Teste de Software

- Teste é uma coisa "óbvia"?
  - "tá pronto "ssora"..."
  - ☹
  - "ué, mas <desculpa > estava funcionando"
    - <desculpa>
      - na minha máquina
      - com o (único) caso que eu testei
      - antes de eu ter acrescentando aquela outra parte
      - ... etc

12

Teste: Exercício



- Um programa recebe pela linha de comando três valores inteiros [x, y, z]. Os três valores devem ser interpretados como sendo os comprimentos dos lados de um triângulo. O programa deve exibir uma mensagem dizendo se o triângulo é equilátero, isóceles ou escaleno.
  - Lembrando:
    - Equilátero – é o triângulo que tem os 3 lados iguais
    - Isósceles – é o triângulo que tem dois lados com medidas iguais
    - Escaleno – é o triângulo que tem os 3 lados com medidas diferentes
- Defina casos de teste: na sua forma mais básica, um conjunto de entradas, condições de execução e saídas esperadas

Por exemplo ...

Entradas			Resultado Esperado
X	Y	Z	
1	1	1	Equilátero
-1	1	1	Erro
1	2	0	Erro
A	1	1	Erro
1	2	3	Escaleno
2	2	4	Isóceles
4	3	3	Isóceles
0	0	0	Erro
2	4	5	Escaleno
2	2	2	Equilátero
0.1	2	3	erro
X	Y	Z	erro

Por exemplo ...

$|b-c| < a < b+c$

Entradas			Resultado Esperado
X	Y	Z	
1	1	1	Equilátero
-1	1	1	Erro
1	2	0	Erro
A	1	1	Erro
1	2	3	ERRO : não forma um triangulo
2	2	4	ERRO : não forma um triangulo
4	3	3	Isóceles
0	0	0	Erro
2	4	5	Escaleno
2	2	2	Equilátero
0.1	2	3	erro
X	Y	Z	erro

Auto-avaliação

1. Do you have a test case that represents a *valid* scalene triangle? (Note that test cases such as 1, 2, 3 and 2, 5, 10 do not warrant a "yes" answer because there does not exist a triangle having these dimensions.)
2. Do you have a test case that represents a valid equilateral triangle?
3. Do you have a test case that represents a valid isosceles triangle? (Note that a test case representing 2, 2, 4 would not count because it is not a valid triangle.)
4. Do you have at least three test cases that represent valid isosceles triangles such that you have tried all three permutations of two equal sides (such as, 3, 3, 4; 3, 4, 3; and 4, 3, 3)?
5. Do you have a test case in which one side has a zero value?
6. Do you have a test case in which one side has a negative value?
7. Do you have a test case with three integers greater than zero such that the sum of two of the numbers is equal to the third? (That is, if the program said that 1, 2, 3 represents a scalene triangle, it would contain a bug.)
8. Do you have at least three test cases in category 7 such that you have tried all three permutations where the length of one side is equal to the sum of the lengths of the other two sides (for example, 1, 2, 3; 1, 3, 2; and 3, 1, 2)?
9. Do you have a test case with three integers greater than zero such that the sum of two of the numbers is less than the third (such as 1, 2, 4 or 12,15,30)?
10. Do you have at least three test cases in category 9 such that you have tried all three permutations (for example, 1, 2, 4; 1, 4, 2; and 4, 1, 2)?
11. Do you have a test case in which all sides are zero (0, 0, 0)?
12. Do you have at least one test case specifying noninteger values (such as 2.5, 3.5, 5.5)?
13. Do you have at least one test case specifying the wrong number of values (two rather than three integers, for example)?
14. For each test case did you specify the expected output from the program in addition to the input values?

Profissional qualificado: 8/14

Teste de Software

- O que significa dizer: "o programa funciona"?
  - Teste de validação
    - Pretende mostrar que o software atende aos seus requisitos
    - Um teste bem sucedido é aquele que mostra que um requisito foi adequadamente implementado
  - Teste de defeitos
    - Testes projetados para descobrir defeitos de sistema
    - Um teste de defeitos bem sucedido é aquele que revela a presença de defeitos em um sistema
- Adequado planejamento de testes
  - Plano de teste
  - Casos de teste

Planejar o que vai ser testado

Utilização de um template

(Test Set #, CT #, Nome do CT, Condições, [Status, Regras relacionadas,Obs])

(1, 1, Caract.Principal 1 – Equilátero, cond1 – ok normal, [- , - , -]
cond2 – lados zerados, [-, -, -])

(1, 2, Caract. Principal 2 – Isósceles, cond1 – ok normal a,a,b []
cond2 – ok normal a,b,a []
cond3 – ok normal a,b,b [])

(1,3, Caract Principal 3 – Escaleno, cond1 – ok normal [])

(1,4, Caract Principal 4 – não triângulo, cond 1 – ok normal [])

Por que testar?

- Vantagens
  - Remover defeitos do produto **ao longo** de todo ciclo de vida
  - Aumento da qualidade e satisfação do cliente
  - Diminuição do re-trabalho
  - Diminuição do custo
    - Baixa qualidade significa altos custos
  - **Porém……**

19

Teste agrega valor mas ...

- teste de software é caro
- existe uma falta de conhecimento sobre a relação custo/benefício do teste
- há carência de profissionais especializados na área de teste
  - Teste automatizado, planejamento de teste, cobertura de teste
- os procedimentos de teste ainda não são bem estabelecidos e amplamente aceitos
- o planejamento adequado das atividades de teste é frequentemente relegado a segundo plano no planejamento do desenvolvimento
  - Fase frequentemente encurtada

20

Papéis e atividades do teste

21

Teste

- Testes devem lidar com
  - Complexidade das aplicações
  - Ambientes
  - Requisitos funcionais e não funcionais
  - Diferentes pontos de vista e diferentes fases do software
  - Níveis de exigência de qualidade
  - Prazos e custos
  - Aspectos éticos

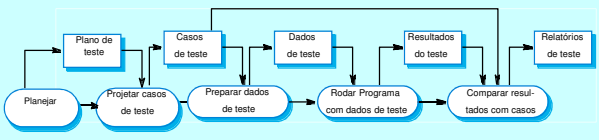
22

Papéis e Responsabilidades

- *Gerente de Teste*
  - normalmente responsável pela definição da política de teste usada na organização, incluindo: planejamento de testes, documentação dos testes, controle e monitoramento dos testes, aquisição de ferramentas de teste, entre outros
- *Engenheiro de Teste* (projetista de teste)
  - responsável principalmente por especificar e projetar os casos de teste
- *Testador*
  - Responsável por executar os casos de teste, e documentar seus resultados
  - Em nível de unidade o testador é freqüentemente o próprio programador

23

Atividades



Adaptado de Sommerville

24

Políticas de teste

- Teste exaustivo é testar para **todas** as possibilidades de execução. Entretanto, na prática nem sempre o teste exaustivo é possível ...
  - Nasa, equipamento médico, controle de reatores nucleares ou aviões
  - Site de compras, redes sociais, sistema de matrículas
- Políticas de teste definem o enfoque a ser usado na seleção dos testes dos sistemas de uma organização:
  - O quê testar ?
  - Como testar? Que nível ou estratégia ou método de teste usar?
    - Dimensões de qualidade, estágio de teste e/ou método de teste
  - Quando testar?

25

Plano de teste

- Testes devem ser planejados
  - Planos de teste são construídos para guiar as atividades de teste ao longo do processo de desenvolvimento de software
- Plano de teste deve especificar
  - O processo de teste
  - Rastreabilidade dos requisitos
  - Casos de teste
  - Cronograma de teste
  - Procedimentos de registro de execução do teste
  - Requisitos de HW e SW para teste
  - conhecimentos/habilidades necessárias
  - Restrições e riscos

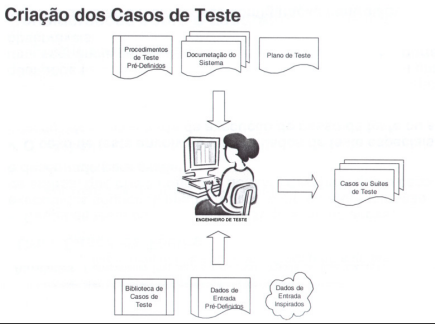
26

Template Plano de Teste - IEEE 829

- Identificador do Plano de Teste
- Introdução
- Itens Testado
- Características a serem testadas
- Características não testadas
- Abordagem
- Critério usado para decidir se o teste passou (pass) ou falhou (fail)
- Artefatos de teste produzidos
- Ambiente necessário
- Responsabilidades
- Schedule
- Riscos

27

Projetos de casos de teste



28

Projeto de casos de teste

- Envolve projetar os casos de teste usadas para testar o sistema
  - complexidade
- O objetivo do projeto de casos de teste é criar um conjunto de testes que sejam efetivos para Teste de Validação e Teste de Defeitos
- Enfoques de Projeto de casos de teste
  - Teste baseado em requisitos (para teste funcional)
  - Casos de teste para exercitar a estrutura
  - Casos de teste para exercitar as propriedades não funcionais

29

Projeto de casos de teste

- São identificadas as características específicas a serem testadas
  - Caso de Teste
    - O que testar
    - Sob quais condições
    - o que é esperado como resultado
  - Especificação dos procedimentos
    - Setup (configuração inicial)
    - Teardow (restauração)

30

Exemplo de caso de teste

Numero do teste: 15

Dependências: 02, 12, 13, 14

Objetivo: verificar se o produto compacta diretórios quando a opção "save folder paths" está desabilitada.

Ambiente:
Hardware: micro PC compatível com pelo menos 5Mb livres no HD
Software:
- **win95 ou superior**
- **diretório c:\tmp criado e vazio**
- **ZipCentral instalado no diretório c:\ZipC**
- **estrutura de diretórios criada no teste 12 presente**

Passos:
1. executar o ZipCentral
2. ativar a opção FileNew
3. observar o resultado esperado 1
4. selecionar o diretório "c:\temp", definir o nome do arquivo como sendo "eca1" e clicar "Ok"
5. observar o resultado esperado 2
6. selecionar o diretório "c:\tclite"
7. clicar o "check box" denominado "include subfolders"
8. clicar "ok"
9. observar os resultados esperados 3 e 4

Resultados esperados:
1. o diálogo "create new archive" deve ser exibido
2. o diálogo "add files and folders to currente file" deve ser exibido
3. a barra de status deve sinalizar a criação do arquivo
4. o arquivo "c:\temp\eca1.zip" deve ter sido criado

31

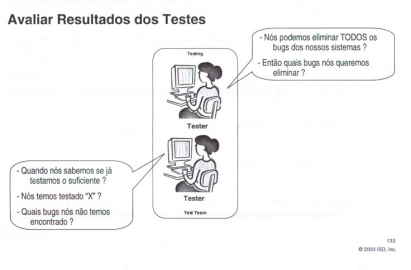
Execução dos Testes

- A execução dos testes começa em nível de unidade até integração e sistema
- Geralmente executado pelo testador
- Teste são executados de acordo com o especificado nos casos de teste definidos no projeto de caso de teste
- Não se "conserta" erros durante o teste: apenas se documentam os problemas encontrados

32

Avaliação e/ou Resultados dos testes

- nesta atividade são relatados todos os resultados e destacadas as discrepâncias encontradas nos resultados.

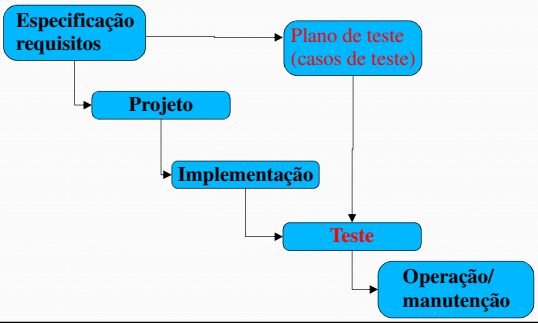


33

Teste no processo de software

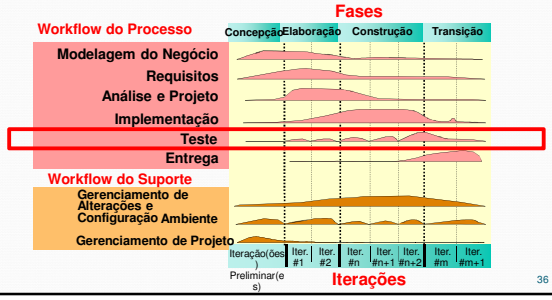
34

Teste no modelo em cascata



35

Teste no modelo incremental

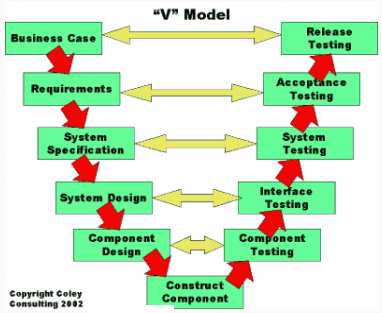


36

Testes no Modelo Incremental

- Testes iterativos e contínuos possibilitam uma medida objetiva do status do projeto
- Inconsistências nos requisitos, projeto e implementação são detectadas mais cedo
- A responsabilidade pela qualidade de trabalho da equipe é melhor distribuída ao longo do ciclo de vida
- Os envolvidos (*stakeholders*) no projeto podem ter evidências concretas do andamento do projeto

Modelo 'V'



Princípios de teste

- Teste é
  - uma fase **PLANEJADA DURANTE** o desenvolvimento do sistema
  - **responsabilidade de todos na equipe**
    - Teste deve ser planejado por analistas, projetistas, programadores e engenheiros de teste ;
    - Teste funcional pode (e deve) ser realizado por desenvolvedores
    - Todos outros testes podem ser realizados por outros perfis na equipe (evitar indução);
- Os melhores testes são aqueles que verificam a **maior gama** de potenciais de erros com o **menor número** de experimentos
- O fim dos testes é ditado geralmente por argumentos **não técnicos**
- Erros são **inevitáveis**
  - Testes não são feitos para validar o código e sim para encontrar os erros que certamente estão em algum ponto dele

O processo de verificação

- Especificação de Requisitos
  - Revisões, inspeções*
    - Verificar se os requisitos são
      - Corretos?
      - Consistentes (não-contraditórios)?
      - Completos?
      - Viáveis?
      - Testáveis?
  - Derivação dos casos de teste (cenários a serem testados)
    - Identificação
    - Especificação

Derivando casos de teste

- "o médico informa a identificação do paciente e os medicamentos prescritos. Os sistema verifica se o paciente tem alguma alergia registrada a um ou mais dentre os medicamentos, alertando em caso positivo. Neste caso, o médico deve confirmar a medicação e fornecer um justificativa, a qual é registrada pelo sistema"

Derivando casos de teste

- Defina um paciente sem alergias a medicamentos. Prescreva medicação para cujas alergias são conhecidas. O alerta não deve ser emitido pelo sistema.
- Defina um paciente com uma alergia a um medicamento. Prescreva esta medicação. O alerta deve ser emitido pelo sistema.
- Defina um paciente com alergia a mais de um medicamento. Prescreva
  - medicações a ambos medicamentos separadamente, verificando que alerta é emitido
  - Duas medicações simultaneamente, verificando que alertas relativos a cada medicamento são emitidos
- Prescreva medicamento que gera alerta, e permita que médico ignore o alerta. Verifique que justificativa foi fornecida pelo médico.
- Etc....

Exercício

O cliente disca para o call center, escutando entao uma mensagem de boas vindas. Tambem são falados todos os filmes à venda, informando para cada um deles a tecla a digitar (e.g. para Matrix, tecle 1). A tecla nove, o não acionamento de uma tecla após um determinado tempo ou a digitação de uma tecla inválida resulta na repetição desta gravação.

Escolhendo o filme, o cliente escuta a sinopse do filme, e em seguida todos os canais e horários onde vais ser exibido, sendo que a cada opção de exibição é associado uma tecla a digitar. A tecla nove, o não acionamento de uma tecla após um determinado tempo ou a digitação de uma tecla inválida resulta na repetição desta gravação. A tecla 0 permite voltar à gravação anterior (escolha do filme).

Escolhendo a exibição, o call center solicita ao cliente a confirmação desta compra através da digitação do numero do cartao de cliente preferencial. Validado este numero, a compra é registrada, e o callcenter emite uma mensagem de agradecimento, e a ligação é desligada.

O usuário tem até três oportunidades para entrar com seu número de sócio, em caso de erros. Após, a operação será cancelada

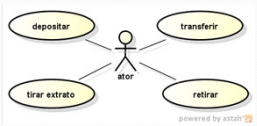
43

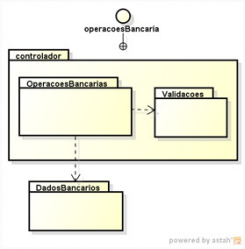
O processo de verificação

- Teste no projeto
  - Casos de teste especificados
  - Revisão do projeto
    - Verificação em relação à especificação
    - Características não funcionais (segurança, desempenho, etc)
  - Projeto visando testabilidade
  - Projeto orientado a testes (TDD – Test-Driven Design)
  - Inspeção

44

Exemplo: aplicação bancária





45

Exemplo: cadastrar Usuário

Sequencia típica de eventos

Usuário informa nome, email único, e senha forte

Sistema **valida** dados, registra novo usuário e confirma cadastramento

Fluxos alternativos

.....

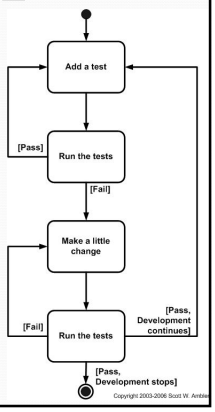
46

TDD

- Estilo de projeto/programação baseado na premissa de que os testes são escritos **antes** do código das classes de "produção"

If it's worth building, it's worth testing.

If it's not worth testing, why are you wasting your time working on it?

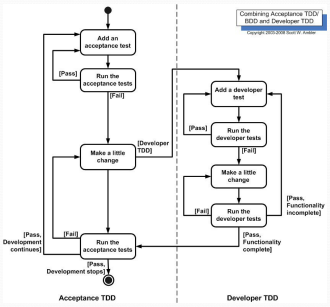


TDD

- Testes de aceitação definidos ANTES da codificação
- Usuário participa ativamente da definição dos testes
- Desenvolvedor sabe perfeitamente o que deve ser feito (o que deve passar no teste de aceitação definido)
- Mais fácil medir o real progresso (% do sistema já está funcionando ?)
- Testes OK aumentam a confiança nas funcionalidades implementadas e dão confiança para refatoração

48

TDD: aceitação vs desenvolvimento



Testes sobre código

- Dados os casos de teste, o objetivo é encontrar os defeitos
  - Desenvolvimento de Código auxiliar
    - Driver
    - stub
  - Execução dos testes
  - Relatório de falhas
  - Reteste (teste de regressão)
  - Planejado e Coordenado pelo **engenheiro de teste**
  - Executado pelo **testador (automatico)**
    - **Precedido pelo programador – teste de unidade**
- Correção do defeito não é parte do teste
  - Uma vez reportado o erro ...
  - Localizar e corrigir o erro
- Instrumentação
  - Ferramentas de depuração
  - Automatização de teste

Dimensões do Teste

Dimensões

- Categoria de verificação
  - Verificação estática vs dinâmica
- Dimensão de qualidade (Tipos de Teste)
  - Os atributos de qualidade que estão sendo o foco do teste
  - Teste Funcional, Teste de segurança, Teste de Stress, Teste de Desempenho, Teste de Usabilidade, etc
- Estágio do teste (Nível de teste, Estratégia de teste)
  - O ponto dentro do ciclo de desenvolvimento em que o teste está sendo executado
  - Teste de Unidade, Teste de Integração, Teste de Sistema, Teste de Aceitação (Homologação)
- Método de teste
  - O objetivo específico do teste
  - técnicas "caixa preta" vs "caixa branca"

Categoria

- Em relação à execução
  - Estático: verificação do artefato (código, especificação, projeto) SEM execução
    - Inspeção (leitura cruzada em equipe com argumentação)
    - Walkthrough ('teste de mesa', simulação)
    - Análise estática de propriedades do código com ferramentas
      - verificação formal de tipos, estruturas de controle, métricas de complexidade
  - Dinâmico: verificação do artefato (frequentemente código) a partir de sua execução sobre conjuntos de dados (casos de teste)
    - Casos de teste são estabelecidos por compromissos de cobertura, prazo e custo
      - Como escolher casos de teste?
      - Como decidir se um resultado é correto ou não?
      - Quando decidir parar os testes?
- O mais executado é o dinâmico, mas os estáticos são igualmente importantes e deveriam ser realizados aos longo de todo o ciclo de vida

Inspeção de Software*

- Envolve pessoas examinando o modelos do sistema ou seu código fonte visando descobrir anomalias e falhas
  - Peer-review, code-review
- Não requer execução do sistema, logo pode ser realizada antes da implementação terminar (ou começar)
- Pode ser aplicada a qualquer representação do sistema (requisitos , modelos de especificação, dados de teste, etc)
- Técnica MUITO efetiva para descobrir erros conceituais
  - Reporte de 30-70% de erros de projeto

* Teste não automatizado ou humano

Análise Estática Automatizada

- Analisadores estáticos são ferramentas (sw) para processamento (análise automática) do código fonte
- Basicamente, fazem um "parse" do programa e tentam descobrir condições potencialmente suspeitas (na nomenclatura OO, "bad smells") e alertá-las para a equipe de teste
  - Variáveis não inicializadas, variáveis/funções não usadas, métricas de acoplamento, etc
- MUITO efetiva como auxílio a inspeção , mas apenas a complementa , não a substitui

Exemplo: LINT

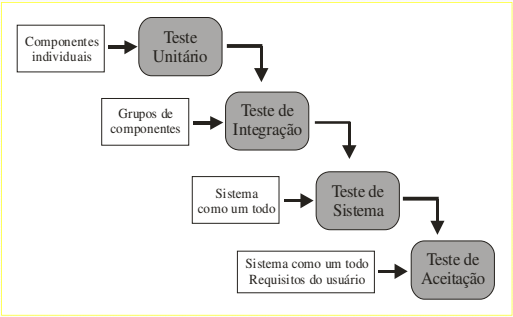
```
138% more lint_ex.c

#include <stdio.h>
printarray (Anarray)
 int Anarray;
{
 printf("%d",Anarray);
}
main ()
{
 int Anarray[5]; int i; char c;
 printarray (Anarray, i, c);
 printarray (Anarray) ;
}

139% cc lint_ex.c
140% lint lint_ex.c

lint_ex.c(10): warning: c may be used before set
lint_ex.c(10): warning: i may be used before set
printarray: variable # of args. lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) :: lint_ex.c(11)
printf returns value which is always ignored
```


Níveis de teste



Também denominado estágio de teste ou estratégia de teste

Teste de Unidade

- **Teste de Unidade (ou Teste Unitário ou Teste de Componente)**
- Seu objetivo é encontrar erros em unidades individuais do sistema, sendo que essas unidades são testadas isoladamente.
- É feito pelo desenvolvedor da unidade (programador)
- Verificação de uma unidade de software
  - Pode ser via **teste funcional**, desenvolvido a partir da especificação das funções previstas para a unidade, ou via **teste estrutural**, desenvolvido a partir da descrição da estrutura do código da unidade.
- Uma unidade pode ser um módulo, uma subrotina, uma *procedure*, uma classe ou até mesmo um programa simples.

Teste de Integração

- O teste de integração tem como foco a arquitetura do sistema, e a integração entre as diferentes unidades que o compõe
  - Envolve o teste dos relacionamentos e interface entre pares de componentes e grupos de componentes do sistema
  - Deve ser feito com as unidades já devidamente testadas (sucede ao Teste de Unidade)
- Normalmente a cargo da equipe de teste ou desenvolvimento
- Facilitado pela modularização do sistema e pela programação defensiva (módulos que testam seus parâmetros de E/S - pré e pós condições)

Teste de Sistema

- Verificação global do sistema após a integração com outros sistemas que deverão operar juntos
- Processo de testar um sistema **pela equipe de testes** para verificar se satisfaz seus requisitos especificados, ou seja, todos os requisitos funcionais e não funcionais.
- Teste do sistema deve ser realizado com todas as partes do sistema já integradas, operando conjuntamente. Portanto, sucede ao Teste de Integração.

Teste de Aceitação

- Teste que envolve verificação do sistema em relação a seus requisitos iniciais feito pelo usuário, ou às necessidades do gestor que solicitou o sistema
- É bastante similar ao teste do sistema; a diferença é que é realizado pelo usuário do sistema (e em ambiente de sistema)
- Teste conduzido para determinar se um sistema satisfaz ou não seus critérios de aceitação e para permitir ao usuário ou gestor determinar se aceita ou não o sistema solicitado
- Requer participação do usuário pois ele é o único que pode validar e aceitar!!

61

Teste de regressão

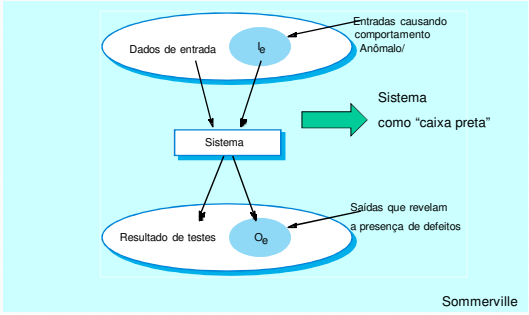
- Estratégia de testes que tenta garantir que:
  - Os defeitos identificados em execuções anteriores dos testes foram corrigidos
  - As mudanças feitas no código não introduziram novos defeitos (ou re-ativaram defeitos antigos)
- Testes de regressão podem envolver a re-execução de quaisquer tipos de testes
- São feitos periodicamente, tipicamente relacionados a interações e testes anteriores

62

Dimensão de Qualidade

Funcional	baseado na especificação Cobertura máxima do comportamento ESPECIFICADO
Estrutural	baseado na estrutura do código Cobertura máxima do comportamento IMPLEMENTADO
Desempenho	Verificação dos objetivos de desempenho de um sistema, estabelecendo propriedades como tempo de resposta e taxas de carga sob certas condições de trabalho e configuração a carga é incrementalmente aumentada requisitos de desempenho vs baseline
Stress	Exercita o sistema além de sua carga máxima projetada força os defeitos a virem à tona
Usabilidade	Verificação facilidades/dificuldades de reconhecimento e uso das funcionalidades disponíveis ao usuário Tempo de aprendizagem, intuitividade
Portabilidade	Observação do comportamento esperado face a mudança de plataforma
outros	Segurança, tolerância a falhas, instalação, etc

Teste Funcional



Sommerville

64

Teste Funcional

- **Teste Funcional (caixa preta)**
  - Visa verificar funcionalidade baseado apenas nos dados de entrada e saída
  - Verificar resultados finais , não importando a estrutura, estados e comportamento internos
  - O termo 'caixa preta' traduz a idéia de uma caixa que **não** permite que seu conteúdo seja visível do lado de fora
- Usa técnicas para determinar os casos de teste

65

Técnicas de Teste Funcional

- Análise do valor limite
  - explora os valores limites do software
- Classes de Equivalência
  - particionar o domínio de entrada em classes
  - Combina dados de diferentes classes
- Tabelas de Decisão
  - Condições
- Baseado em estados
  - Explora as condições próprias a cada estado

66

Análise de valor limite

- Critério de seleção que identifica valores nos limites das classes de equivalência
  - Obs: variações quanto a usar um valor médio
- Exemplos:
  - Valor
    - valor mínimo (máximo) igual ao mínimo (máximo) válido
    - uma unidade abaixo do mínimo
    - uma unidade acima do máximo
  - Arquivo
    - arquivo vazio
    - arquivo maior ou igual à capacidade máxima de armazenamento

67

Análise do valor limite

- Exemplo: entrar dois valores no intervalo de [1..100]
  - Nro de entradas
    - 1 entrada
    - 2 entradas
    - 3 entradas
  - Intervalo
    - 0
    - 1
    - 50
    - 100
    - 101

68

Classe de Equivalência

- **O domínio de entrada (ou saída) do programa/função é dividido em um número finito de partições (ou classes) de equivalência**
  - **supõe-se que dados pertencentes a uma partição revelam as mesmas falhas**
  - **partições válidas e inválidas são consideradas**
- **Geração de testes: selecionar um ou mais dados de cada partição**
- **Critério de cobertura: cada partição deve ser considerada ao menos 1 vez**

69

Classes de Equivalência : Passos

- Decompor o programa em funções
- Identificar as variáveis que determinam o comportamento de cada função
- Particionar os valores de cada variável em classes de equivalência (válidas e inválidas)
- Especificar os casos de teste:
  - eliminar as classes impossíveis ou os casos desinteressantes
  - selecionar casos de testes cobrindo as classes válidas das diferentes variáveis
  - para cada classe inválida escolha um caso de teste que cubra 1 e somente 1 de cada vez

70

Exemplo

- Função:
  - Considere uma função que aceita como entradas de 4 a 6 valores inteiros de 2 dígitos maiores do que 10.
- Identificação das variáveis de entrada e das condições que estas devem satisfazer:
  - nro_entradas ∈ [4, 6]
  - valor ∈ [10, 99]

Exemplo extraído de material de Profa. Eliane Martins - Unicamp

71

Exemplo

- Determinação das classes de equivalência:

Variável	Classes Válidas	Classes Inválidas
nro_entradas	C1. 4 ≤ nro_entradas ≤ 6	C3. nro_entradas < 4 C4. nro_entradas > 6
valor	C2. 10 ≤ valor ≤ 99	C5. valor < 10 C6. valor > 99

72

Exemplo

- Casos de teste:
  - selecionar casos de testes cobrindo as classes válidas das diferentes variáveis

nro_entradas	C1	C2	C3	C4	C5	C6
4 ... 6	X					
< 4						
> 6						
valor						
10 ... 99		X				
< 10						
> 99						

nro_entradas	valores
5	11, 12, 45, 78, 95

Exemplo

- Casos de teste:
  - para cada classe inválida escolha um caso de teste que cubra 1 e somente 1 de cada vez

nro_entradas	C1	C2	C3	C4	C5	C6
4 ... 6	X					
< 4			X			
> 6						
valor						
10 ... 99		X				
< 10						
> 99						

	nro_entradas	valores
1.	5	11, 12, 45, 78, 95
2.	3	11, 12, 45

Exemplo

- Casos de teste:
  - para cada classe inválida escolha um caso de teste que cubra 1 e somente 1 de cada vez

nro_entradas	C1	C2	C3	C4	C5	C6
4 ... 6	X					
< 4			X			
> 6				X		
valor						
10 ... 99		X				
< 10					X	
> 99						X

	nro_entradas	valores
1.	5	11, 12, 45, 78, 95
2.	2	11, 12
3.	8	11, 12, 45, 78, 95, 67, 77, 54
4.	5	5, 11, 12, 45, 6
5.	5	110, 45, 78, 340, 95

Exercício

- Considere o teste do procedimento: Valida_Nova_Senha que recebe como entrada uma senha e valida-a conforme as seguintes regras:
  - uma senha deve ter de 6 a 10 caracteres
  - o primeiro caracter dever ser alfabético, numérico ou um "?"
  - os outros caracteres podem ser quaisquer, desde que não sejam caracteres de controle
  - a senha não pode existir em um dicionário

Defina as classes de equivalência

Defina os casos de teste

Exercício

Variáveis	Válidas	Inválidas
tamanho	C1. tamanho ∈ [6, 10]	C7. tamanho < 6
		C8. tamanho > 10
1° caracter (Car1)	C2. Car1 alfabético	C9. Car1 ∈ {caract. controle}
	C3. Car1 numérico	C10. Car1 ∉ {letras, dígitos, ?}
	C4. Car1 é "?"	
outros caract (Outro)	C5. Outro ∉ {caract. de controle }	C11. Outro ∈ {caract. de controle}
status	C6. Senha ∉ dicionário	C12. Senha ∈ dicionário

Exercício

	casos de teste									casos de teste
	1	2	3	4	5	6	7	8	9	
1										1.
2										2.
3										3.
4										4.
5										5.
6										6.
7										7.
8										8.
9										9.
10										
11										
12										

classes de equivalência

Partição de Equivalência e Valor Limite

- Bastante relacionadas
  - O valor limite auxilia a encontrar partições de valores válidos/invalidos
  - Os valores limite de uma partição são seu máximo e seu mínimo
  - Procurar fazer o mínimo de combinações que leva à maior cobertura de teste
- Limitação
  - Consideram os casos em isolado
  - Algumas combinações levam a alguns casos específicos interessantes de serem testados

79

Software de suporte ao processo

- Ambientes gerenciadores
  - Gerenciadores/editores de planos de teste
    - ex.: IBM/Rational Test Manager, HP/Mercury Quality Center
  - *Acompanhamento de bugs (bug tracking)*
    - *Ex.: BugZilla,* IBM/Rational *ClearQuest,* HP/Mercury *Quality Center*
  - *Ambientes de controle de atividades*
    - *Ex: TRAC, MS TFS*
- Execução automática de testes
  - *Capture-playback*
  - *Scripts*
  - *Frameworks*
  - *Ex.: TestRunner, Rational Robot, família XUnit (e.g. JUnit, VBUnit),* HP/Mercury *QuickTest, TestComplete, Autotest*

80

Idéias do Teste Automatizado

- Escrever testes faz parte do desenvolvimento, e parte integral do projeto/implementação
  - Testar cedo e testar sempre
- Permite experimentar diferentes idéias de projeto (sem quebrar o que estava funcionando):
  - Inicie com "o mais simples que possa funcionar"
  - Refine o projeto através do uso de padrões e aplicação de refatorações
- Tentar quebrar o ciclo: "Mais pressão, pule o teste"
  - Menos tempo com atividades de depuração
  - Regressão
  - Integração contínua
- Requisitos "vivos"

Exemplo : JUnit

- Junit é um framework de testes de regressão desenvolvido por Erich Gamma e Kent Beck
  - Adaptado a várias linguagens : família XUnit
- JUnit é uma ferramenta que suporta a criação e execução de testes de unidade
- JUnit estrutura os testes e provê mecanismos para executá-los automaticamente
  - Asserções
  - Rodar testes
  - Agregar testes (suites)
  - Mostrar resultados

http://www.junit.org

Asserções disponíveis no Junit

Table 1.2 The JUnit class `Assert` provides several methods for making assertions.

Method	What it does
`assertTrue(boolean condition)`	Fails if `condition` is false; passes otherwise.
`assertEquals(Object expected, Object actual)`	Fails if `expected` and `actual` are not equal, according to the `equals()` method; passes otherwise.
`assertEquals(int expected, int actual)`	Fails if `expected` and `actual` are not equal according to the `==` operator; passes otherwise. There is an overloaded version of this method for each primitive type: `int`, `float`, `double`, `char`, `byte`, `long`, `short`, and `boolean`. (See Note about `assertEquals()`.)
`assertSame(Object expected, Object actual)`	Fails if `expected` and `actual` refer to different objects in memory; passes if they refer to the same object in memory. Objects that are not the same might still be equal according to the `equals()` method.
`assertNull(Object object)`	Passes if object is `null`; fails otherwise.

Exemplo

```
public class Calc
{
        public long add(int a, int b) { return a+b; }
}
```

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class CalcTest
{ @Test
        public void testAdd()
                { assertEquals(long(5), new Calc().add(2, 3)); }
}
```

84

UML e Teste

- *UML 2.0 Testing Profile Specification* (U2TP)
  - Notação com conceitos específicos para descrição de casos de teste em alto nível
  - Notação standard e gráfica
  - A partir da UML, é possível gerar código para automatização de testes
    - Ex: JUnit

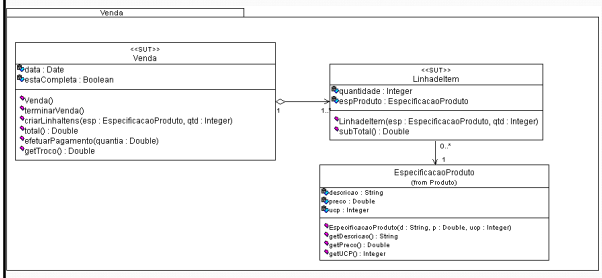
U2TP

- *Arquitetura de Teste* - conceitos relacionados a estrutura e configuração de teste.
- *Dados de Teste* - conceitos para dados usados em procedimentos de teste.
- *Comportamento de teste* - conceitos relacionados aos aspectos dinâmicos dos procedimentos de teste.
- *Tempo de Teste* - conceitos quantificados por tempo para procedimentos de teste.

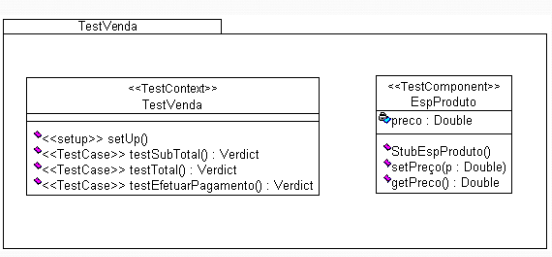
U2TP

Conceitos de Arquitetura de teste	Conceitos de Comportamento de teste	Conceitos de Dados de Teste	Conceitos de Tempo
Sut	*TestObjective*	*Wildcards*	*Timer*
TestComponent	*TestCase*	*DataPool*	*TimeZone*
TestContext	*Defaults*	*DataPartition*	
TestConfiguration	*Validation action*	*DataSelector*	
TestControl	*Verdicts*	*Coding Rules*	
Arbiter			
Scheduler			

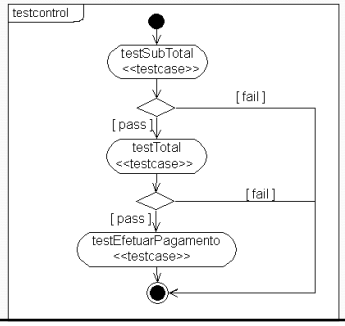
U2TP: Exemplo



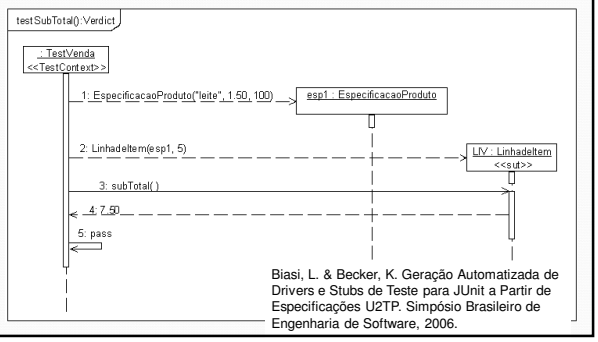
U2TP: Exemplo



U2TP: Exemplo



U2TP Exemplo



Biasi, L. & Becker, K. Geração Automatizada de Drivers e Stubs de Teste para JUnit a Partir de Especificações U2TP. Simpósio Brasileiro de Engenharia de Software, 2006.

Conclusões

- Teste é parte central da avaliação de qualidade
- Testes/Inspeções se aplicam a diferentes artefatos do ciclo de vida
  - verificação contínua e incremental
- Processo de teste não é trivial
  - Planejamento cuidadoso
  - Casos de teste
  - Técnicas apropriadas
  - Cobertura dos testes
  - Não garante a ausência de erros
- Ainda carecemos de processos, procedimentos e notações apropriadas para apoiar o processo de teste
- Automação de testes vem se tonando fundamental
  - Mas resolve apenas uma parte do problema

92

Para saber mais …

- Leitura fortemente recomendada
  - Sommerville, I. Engenharia de Software. Pearson.
    - Capítulos 19 e 20.
- Leituras complementares para interessados
  - Pezzè, M.; Young, M. Software Testing and Analysis – Process, principles and Techniques, Wiley, 2008. (tem tradução pela Bookman)
  - Myers, G. The Art of Software Testing John Wiley and Sons , New York, 1979.
  - Beck, K. Test-Driven Development by Example, Addison Wesley, 2003

93