Fases de Teste:
Testes de Sistemas
e Regressão

Criado: junho/2006

Últ.modificação: outubro/2011

Tópicos

- Objetivos
- Fontes de informação para os testes
- Testes de requisitos funcionais
- Testes de requisitos não-funcionais (atributos de qualidade):
  - Testes de desempenho
  - Testes de robustez

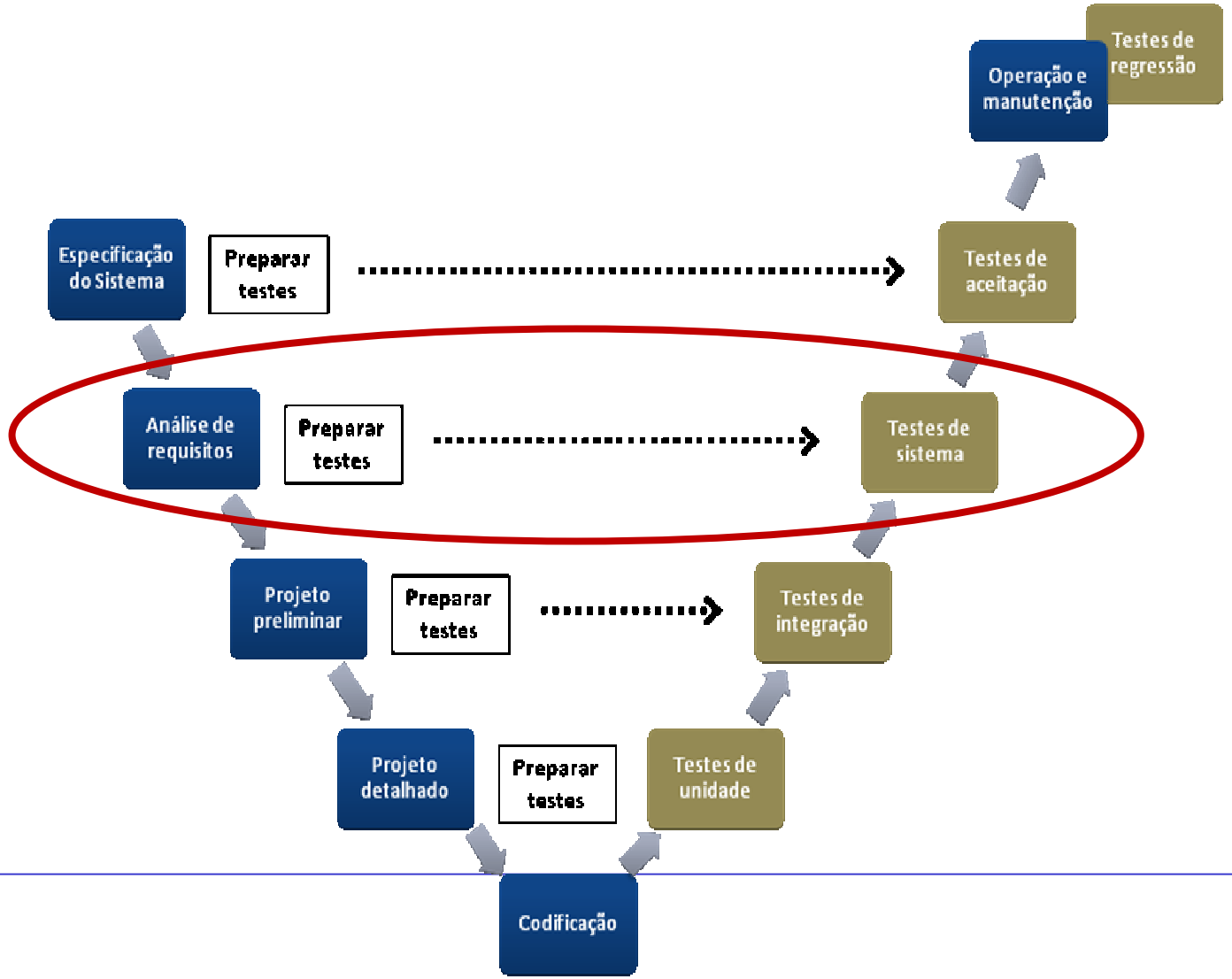
Referências

- Leonardo Molinari. "*Testes de Software: Produzindo Sistemas Melhores e Mais Confiáveis*", Editora Érica Ltda, São Paulo 2003.
- Arturo H.T. Zenteno. "*Processo de Desenvolvimento e Testes para Aplicações SIG Web*". Trabalho Final de Mestrado Profissional. Jan/2006.
- Adriano L.C. Leite, Jane E. Morales. "*Ferramentas CASE – JMeter*". Trabalho apresentado na disciplina INF308 – out/2005.
- Henrique Madeira. "*Fault Injection*". Tutorial apresentado em 2004.
- Regina Lúcia de Oliveira Moraes, Eliane Martins, Elaine Cristina Catapani Poletti, Naaliel Vicente Mendes. "*Using Stratified Sampling for Fault Injection*". Apresentado no Latin-American Symposium on Dependable Computing (LADC) 2005, Salvador, BA, Brasil.
- R. Moraes, R. Barbosa, J. Durães, N. Mendes, E. Martins, H. Madeira. "*Do injected component interface faults represent software bugs?"*. A ser apresentado na European Dependable Computing Conference (EDCC), 2006, Coimbra, Portugal.
- R.S. Pressman. Engenharia de Software. 6ª edição, 2005, McGraw Hill, c. 13.6.

Mais referências

- C. Cachin, J. Camenisch, M. Dacier, Y. Deswarte, J. Dobson, D. Horne, K. Kursawe, J.C. Laprie, J.C. Lebraud, D. Long, T. McCutcheon, J. Muller, F. Petzold, B. Pfitzmann, D. Powell, B. Randell, M. Schunter, V. Shoup, P. Verissimo, G. Trouessin, R.J. Stroud, M. Waidner, and I. Welch, ―Malicious- and Accidental-Fault Tolerance in Internet Applications: Reference Model and Use Cases, ‖ LAAS report no. 00280, MAFTIA, Project IST-1999-11583, p. 113, Aug. 2000.
- PROTOS - Security Testing of Protocol Implementations. Obtained in Mai/2008 at: http://www.ee.oulu.fi/research/ouspg/protos/.
- N. Neves, J. Antunes, M. Correia, P. Veríssimo, R.Neves. ―Using Attack Injection to Discover New Vulnerabilities ‖ . In: Proc. of the International Conference on Dependable Systems and Networks (DSN), 2006, pp 457-466.
- Herbert H. Thompson, James A. Whittaker, Florence E. Mottay. ―Software security vulnerability testing in hostile environments ‖ . ACM Symposium on Applied Computing (SAC) 2002: 260-264. Madrid, Spain.
- Ricardo D. da Silva. Metodologias para Validação de Segurançaa NIST e OSSTMM Aplicadas ao Sistema de venda de Bilhetes pela Internet. Trabalho apresentado como parte da disciplina MO409-2009.

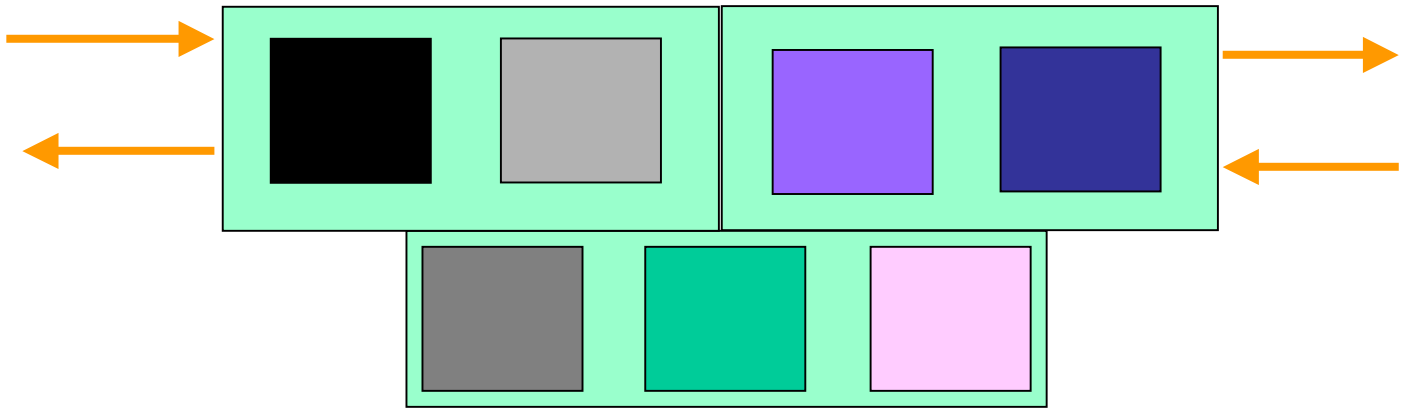
Testes no Processo de Desenvolvimento



Objetivos

- Combinação de testes que tem por objetivo:
  - revelar falhas de sistema
  - demonstrar que o sistema em teste implementa os requisitos funcionais e não funcionais

  ➲ o sistema foi construído corretamente ?



Fontes de informação para os testes

- Especificação de requisitos.
- Protótipo, layouts ou modelos da IU.
- Políticas da organização implementadas como objetos de negócio, “stored procedures” ou “triggers”.
- Características do produto descritas na literatura.
- Características e procedimentos descritos na documentação, telas de ajuda ou assistentes de operação (“wizards”).
- Padrões.

Especificação deve ser:
- **completa**
- **consistente**
- **precisa**

⇨ **testável**

Testes dos requisitos funcionais

- Visam verificar se as funcionalidades especificadas foram devidamente implementadas

⇩

- Uso de métodos de testes caixa-preta :
  - partição de equivalência
  - valores-limite
  - tabela de decisão / grafo causa-efeito
  - modelos de estado
  - diagramas de casos de uso
  - cenários
  - ...

Testes dos requisitos não funcionais

- Visam determinar se a implementação do sistema satisfaz aos requisitos não funcionais
- Tipos de testes:
  - configuração e compatibilidade
  - desempenho
  - estresse
  - usabilidade
  - tolerância a falhas
  - segurança
  - ...

Testes de configuração e compatibilidade

- Verificam se a implementação é capaz de executar nas diferentes configurações do ambiente alvo que foram especificadas
- Verificam se a implementação é capaz de interoperar com sw e hw conforme especificado:
  - sistemas já existentes
  - plataformas específicas de hw
  - diferentes (versões de)sistemas operacionais
  - diferentes interfaces gráficas (X-Windows, Microsoft Windows)
  - diferentes API e IDL
  - diferentes SGBD

Testes de desempenho

- Visam determinar se implementação satisfaz aos requisitos de desempenho especificados:
  - configuração de rede
  - tempo de CPU
  - limitação de memória
  - carga do sistema
  - taxa de chegada de entradas

  ➲ esses requisitos devem ser descritos de forma testável

  ex.: nº de transações/seg ou tempo de resposta em seg, mseg

O sistema é testado em condições **reais** de operação.

Variações dos testes de desempenho

- Testes de carga
  - geralmente associados com sistemas transacionais
  - usam simuladores de carga para geração de múltiplas transações/usuários simultaneamente
- Testes de volume
  - geralmente usados para sistemas “batch”
  - consistem na transmissão de um grande volume de informações quando o sistema está com a carga normal ou
  - uso de arquivos grandes (maior tamanho possível) ou de grande número de arquivos

Teste de estresse

- Visa ir além dos limites do sistema, seja em nº de usuários simultâneos, seja em volume de dados, seja em nº de processos ou de transações:
  - verificar se o sistema não apresenta um comportamento de risco quando submetido a carga elevada e com um ou mais recursos saturados.
- Importância:
  - muitos sistemas apresentam **comportamento de risco** nessa situação
  - falhas detectadas são sutis
  - correções desse tipo de falha podem requerer retrabalho considerável

Objetivos

- **Eliminação de falhas** → verificação
  - Resultado: veredicto (Sucesso, Defeito)
- **Previsão de Falhas** → medição
  - Cobertura de falhas:
    #falhas reveladas / #falhas injetadas
  - Nº de defeitos ocorridos
  - ...

Medidas indiretas de confiabilidade

Medidas de confiabilidade

Confiabilidade: capacidade de atender a especificação, dentro de condições definidas, durante certo período de tempo e condicionado a estar operacional no início do período

Taxa de defeitos (*failure rate*)	número esperado de defeitos em um dado período de tempo, assumido como um valor constante durante o tempo de vida útil do componente
MTTF (*Mean Time to Failure*)	tempo esperado até a primeira ocorrência de defeito
MTTR (*Mean Time to Repair*)	tempo médio para reparo do sistema
MTBF (*Mean Time Between Failures*)	tempo médio entre defeitos do sistema

Testes de robustez

- Objetivo:
  - verificar a capacidade de um sistema, ou componente, de funcionar de forma correta (ou ao menos, aceitável) em presença de falhas ou de condições ambientais estressantes
  - Em outros termos:
    - Testes de sistema ou componente em presença de falhas externas

Testes de robustez: por que?

- Dependência crescente de sistemas computacionais
- Serviços dependem cada vez mais do software
- Mau-funcionamento desses sistemas:
  - Podem ter conseqüências catastróficas

…

http://publico.pt/Tecnologia/computador-infectado-pode-ser-uma-causa-da-queda-do-aviao-que-matou-154-pessoas-em-espanha-1452553

Acidente da Spanair

Computador infectado pode ser uma causa da queda do avião que matou 154 pessoas em Espanha

23.08.2010 - 16:39 Por PÚBLICO

Votar ☆☆☆☆☆ | 1 votos ★★★★☆ 3 de 3 notícias em Tecnologia « anterior

O computador da companhia aérea Spanair onde eram registadas as avarias dos aviões estava infectado com *software* malicioso, o que fez com que os técnicos não tivessem introduzido informação sobre as falhas detectadas no avião.

O avião despenhou-se após a descolagem, no aeroporto de Barajas (Reuters (arquivo))

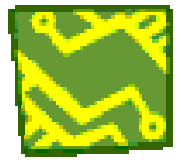
Testes de robustez: por que?

- Dependência crescente de sistemas computacionais
- Serviços dependem cada vez mais do software
- Mau-funcionamento desses sistemas:
  - Podem ter conseqüências catastróficas ou
  - Podem afetar milhares de pessoas

- THOUSANDS of iPhone users overslept and many were late for work and flights yesterday after a -software glitch cancelled its alarm.
- The iPhone's non-recurring alarms stopped for two days after clocks struck midnight to usher in 2011.
- The bug follows similar problems when clocks went back at the end of British summer time in October.
- Read more: http://www.mirror.co.uk/news/most-popular/2011/01/03/thousands-of-iphone-users-hit-by-software-failure-which-cancelled-the-phone-s-alarms-115875-22822789/#ixzz1WH6VMUKv

Injeção de falhas

- Histórico
  - Anos 70, visava testar componentes de hw
    - Introdução de defeitos de hw



- O que é
  - Técnica de validação de software que consiste em observar o funcionamento de um sistema em presença de falhas ou erros.
- Objetivos:
  - Verificação – remoção de falhas de software no sistema em teste.
  - Medição – obtenção de medidas de atributos de qualidade: confiabilidade, disponibilidade, entre outras.
- Técnica muito utilizada nos testes de robustez

Principais técnicas

- Injeção por hardware:
  - Pinos de componentes
  - Radiação íons pesados ou eletromagnéticas
  - Perturbações na fonte de alimentação
  - Entre outros

Complexidade do hw
Baixa controlabilidade
Baixa observabilidade do efeito das falhas
Grande custo de desenvolvimento
Baixa portabilidade

- Simulação de falhas:
  - Geralmente são falhas que afetam um modelo de componentes de hw: circuitos, portas lógicas, etc.

Modelos complexos
Esforço de desenvolvimento é alto
Tempo de execução dos experimentos é alto

- Injeção por software:
  - Software que injeta falhas
  - Boundary scan

Injeção de falhas por software

- Princípio:
  - A execução do sistema alvo é interrompida de alguma forma:
    - Execução em “trace mode”, temporizador, instrução executada, ...
  - Uma rotina de injeção de falhas é executada, emulando a ocorrência de falhas pela introdução de erros em diferentes partes do sistema:
    - Registradores, memória, variáveis, parâmetros, mensagens, ...
  - A execução do sistema é retomada.
  - O comportamento do sistema é observado para determinar a manifestação do erro:
    - Fronteira de componentes, de sistema, tratadores de erro/exceção.

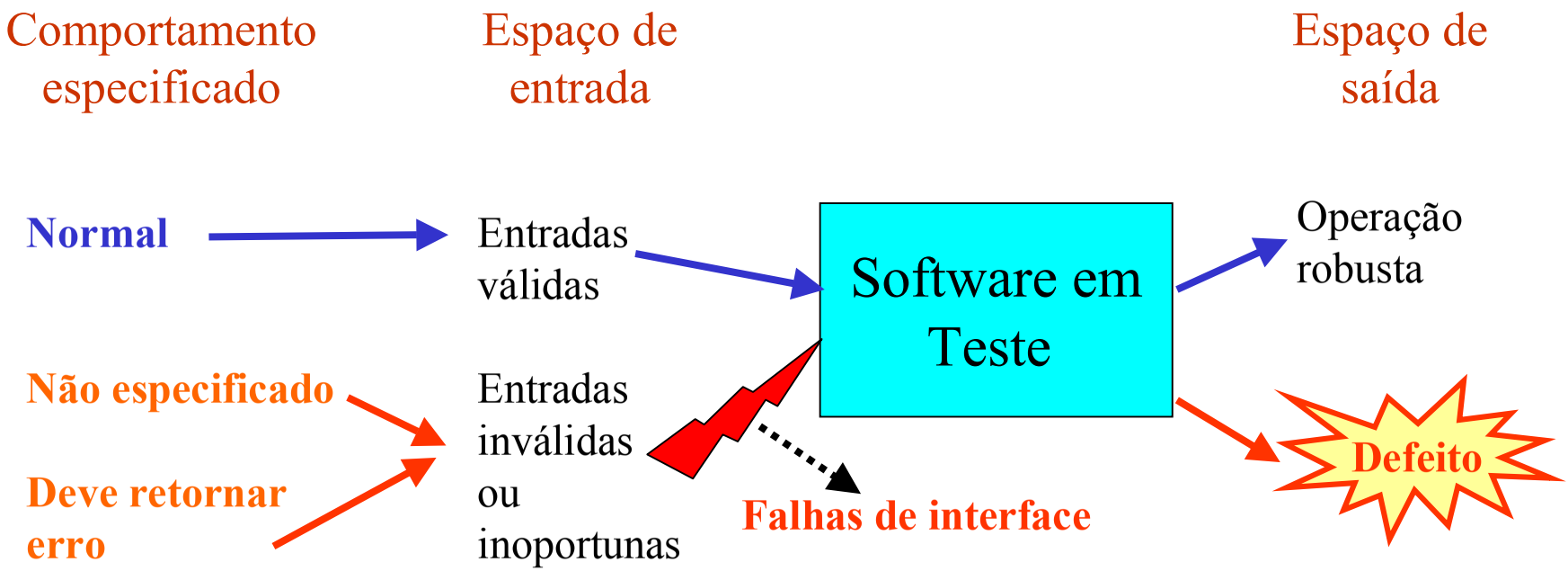
Vantagens e limitações

- Vantagens:
  - Pouco afetada pela complexidade do sistema alvo
  - Baixa complexidade
  - Baixo custo de implementação de injetores
  - Maior portabilidade
  - Não tem interferência física com o sistema alvo, ou seja, não há risco de danificá-lo
- Limitações
  - Difícil testar dispositivos periféricos
  - Impacto dos injetores no sistema alvo é alta

Algumas ferramentas

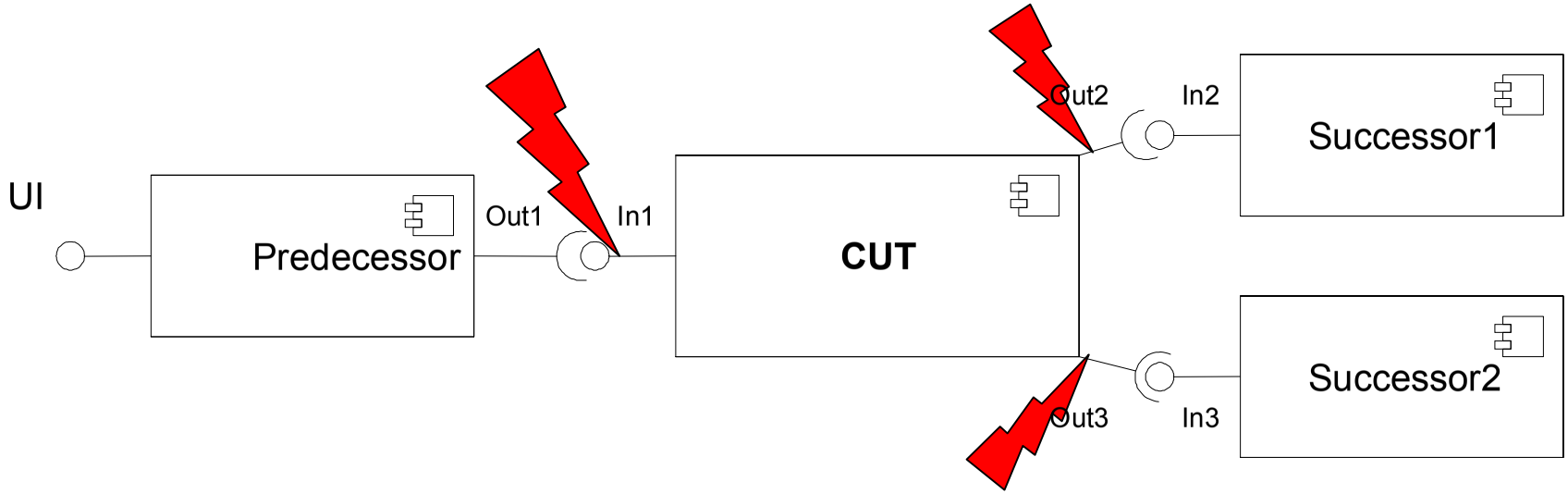
- Existem diversas ferramentas:
  - Orchestra, Ftape, ComFirm, ...
- Xception (Univ. Coimbra → Critical):
  - Emula falhas de hw
  - Usa capacidade embutida de monitoração e debugging existente em processadores modernos (PowerPC, Pentium, ...) para injetar falhas com baixa interferência.
  - Injeta falhas em unidades aritméticas (inteiros e ponto flutuante), barramentos de dados e de endereços, registradores, memória.

Injeção de falhas nos testes de robustez



[base: Koopman99]

Testes de Robustez em Sistemas Baseados em Componentes



Injetar falhas nas interfaces entre o componente e o resto do sistema:
- O componente é robusto com relação às falhas de outros componentes?
- O sistema é robusto com relação às falhas do componente escolhido?
- Qual o risco de usar o componente no sistema?

Abordagens e (algumas) Ferramentas (1)

- MAFALDA (LAAS-CNRS)
  - Microkernel Assessment by Fault injection AnaLysis and Design Aid
  - Teste de núcleo de sistemas operacionais (e.g. LynxOS)
  - Injeta falhas nos parâmetros de chamadas ao sistema

- BALLISTA (Carnegie-Mellon)
  - Abordagem e ferramenta
  - Combina injeção de falhas e testes (valores-limites)
    - Valores válidos e inválidos nos limites
    - Valores escolhidos de um BD de valores pré-definidos
    - Valores variam de acordo com o tipo do dado
  - Usado nos testes de sistemas operacionais
    - Diferentes versões do Unix; Windows

Abordagens e (algumas) Ferramentas (2)

- Fuzz:
  - Entradas geradas aleatoriamente usando teclado e/ou mouse
  - Usada nos testes de aplicações Unix via linha de comando (1990)
  - Testes do X-Window (1995)
  - Testes do Windows (2000)
  - Colapso (*crash*) de 40% das aplicações testadas
- Jaca (Unicamp):
  - Injeção de falhas de interface:
    - Injeta em parâmetros e retorno de funções
  - Usa reflexão computacional para afetar o comportamento de programas Java.
  - Usado nos testes de um gerenciador de banco de dados OO
- ...

Exemplo modelo de falhas - Ballista

API: write(int filedesc, const void *buffer, size_t nbytes)

Tipos de dados	descritor de arquivos	buffer de memória	tamanho
Valores de teste	FD_CLOSED FD_OPEN_READ FD_OPEN_WRITE FD_DELETED FD_EMPTYFILE ...	BUF_SMALL_1 BUF_LARGE_512M BUF_HUGE_2G BUF_NULL BUF_16 ...	SIZE_1 SIZE_ZERO SIZE_NEG SIZE_MININT SIZE_MAXINT ...

Caso de teste: write(FD_CLOSED, BUF_NULL, SIZE-NEG)

(inspirado em Koopman2008)

Mais valores do modelo Ballista

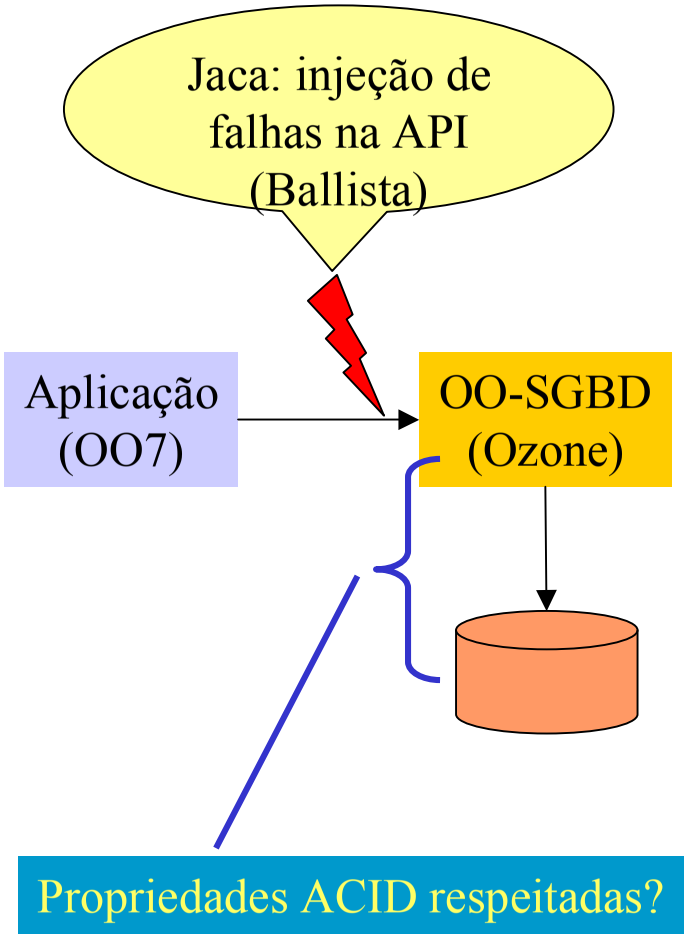
Tipo do dado	Valores
Inteiro	0, 1, -1, MaxInt, MinInt
Real	0., 1., -1., DblMin, DblMax
Boolean	Inversão de estado (V→ F, F → V)
String	Null, string do tamanho da memória virtual, string com caracteres especiais (fim de arquivo, formatação, etc)
Descritor de arquivo (tipo inteiro)	0, 1, -1, MaxInt, MinInt descritor de: arquivo aberto para leitura, arquivo aberto para escrita, arquivo vazio, arquivo apagado após o descritor ter sido atribuído

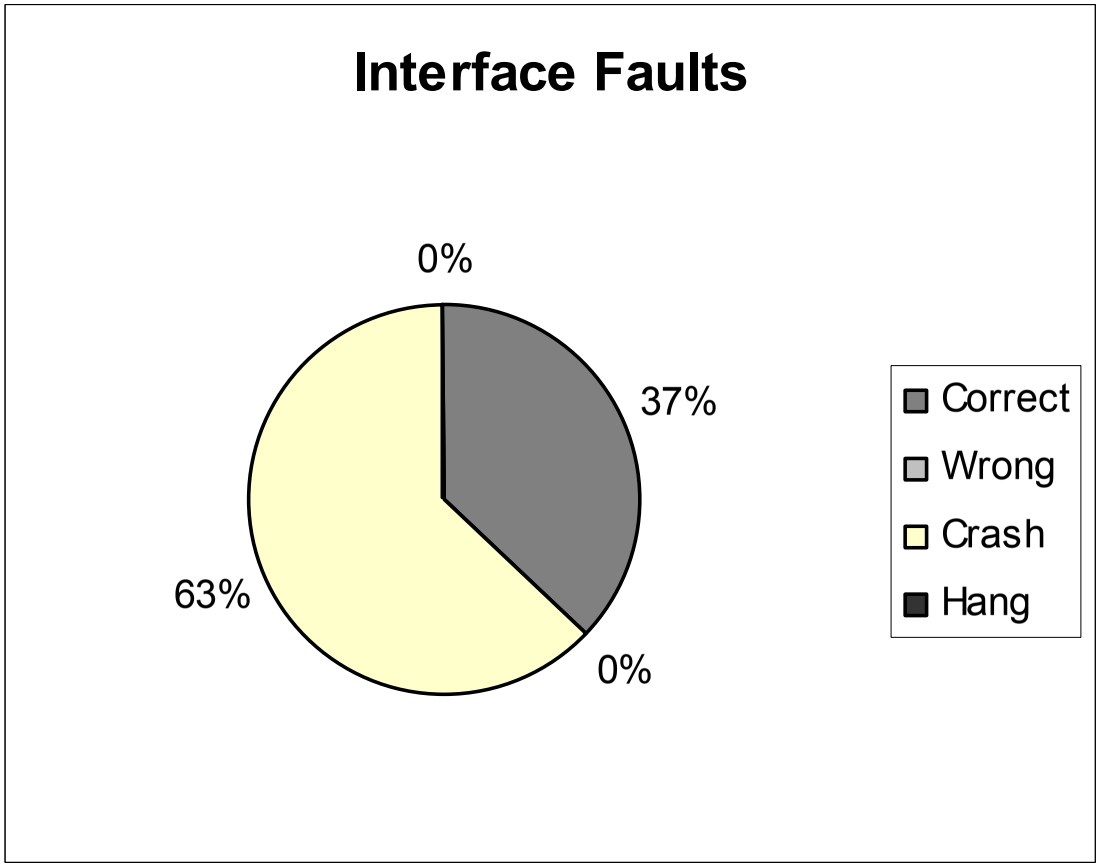
Fonte: Projeto Ballista - http://www.ece.cmu.edu/~koopman/ballista/

Ballista: classificação da não-robustez

- Escala **CRASH** para classificar os defeitos de robustez :
  - **C**atastrófico:
    - S.Op. é corrompido; a máquina "cai" ou reinicia
  - **R**einicialização (*Restart*):
    - A aplicação fica bloqueada e precisa ser abortada
  - **A**borto:
    - A aplicação terminou anormalmente (abortou por si)
  - **S**ilêncio:
    - Nenhum erro (exceção) foi assinalado, quando deveria ter sido
  - Obstrução (***H****indering*):
    - O código de erro retornado não é o correto

Exemplos de resultados





Mais exemplos de resultados

Saídas observadas	Total
Término normal	200
Exceção levantada pela aplicação	4
Exceção levantada pelo gerenciador de BD	—
Término normal, mas integridade do BD violada	5
Término anormal e integridade do BD violada	1
Total	**210**

Testes de segurança

- Visam verificar a capacidade do sistema de impedir acesso não autorizado, sabotagem ou outros ataques intencionais
- Testam a capacidade do sistema de resistir a ataques
  - Quem realiza os testes deve “pensar” como um atacante
- Mas ...

  O que é segurança?

SEGURANÇA

Disponibilidade → **Serviços e recurso do sistema estão prontos para serem usados**

Confidencialidade → **Ausência de acesso não autorizado à informação**

Integridade → **Ausência de adulteração da informação**

- Outras propriedades:
  - **Privacidade**: prevenção de interceptação (não autorizada) de informação
  - **Autenticação:** identificação e validação de usuário para acesso à informação
  - **Irretratabilidade** (*non-repudiation*): prevenção da negação de serviço de envio ou recepção de informação
  - ...

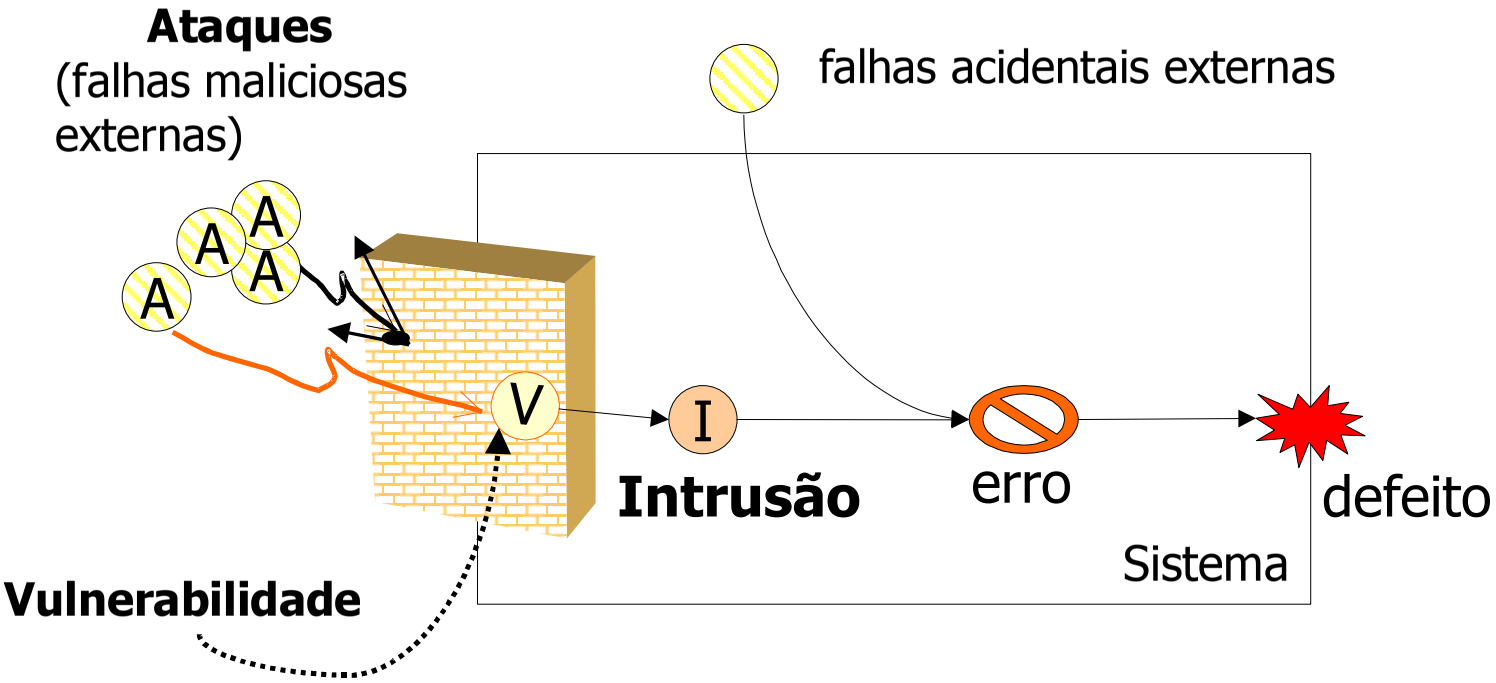
Ameaças à segurança - 1

- Vulnerabilidade
  - *É uma falha (maliciosa ou não) introduzida durante o desenvolvimento e que pode ser usada por atacantes ter acesso ao sistema ou a informações*
  - Causas para seu aparecimento são várias:
    - Complexidade
    - Senhas fracas
    - Deficiências no gerenciamento de senhas e privilégios
    - Deficiências do Sistema Operacional
    - Falhas de software
    - ...

Ameaças à segurança -2

- Vulnerabilidade
- Ataques:
  - São atividades externas maliciosas que violam propriedades de segurança (ex.: confidencialidade, integridade,...).
  - Várias classificações:
    - Quanto às propriedades de segurança visadas
    - Quanto à forma de realizar

Resumo: ameaças à segurança



Ameaças à segurança -3

- Vulnerabilidade
- Ataques

- Intrusão = ataque + vulnerabilidade.

Técnicas de Ataque

- **Engenharia social**: consiste em enganar ou manipular as pessoas para que estas realizem ações que favoreçam ao atacante ou forneçam informações sigilosas
  - Ex.: mensagens de phishing, em que o atacante se passa por outra pessoa ou por empresa
- **Comprometimento de páginas**: infecção de páginas Web
- **Anúncios maliciosos**: anúncios que direcionem o usuário para páginas com cnteúdo malicioso
- **Manipulação de mecanismos de busca**: faz com que mecanismos de busca coloquem em posição alta a página com conteúdo malicioso

Como proteger o software? (1)

- Desenvolvimento de mecanismos de defesa, tais como:
  - *Firewalls*
  - Sistemas de Detecção de Intrusão
  - Criptografia
  - Controle de software
    - Ex.: limitação de acesso a recurso
  - Controle de hardware
    - Ex.: autenticação usando smartcards
  - Políticas de segurança
    - Ex.: alteração periódica de senhas
  - Controle físico
  - ...

Como proteger o software? (1)

- Desenvolvimento de mecanismos de defesa, tais como:
  - *Firewalls*
  - Sistemas de Detecção de Intrusã
  - Criptografia
  - Controle de software
    - Ex.: limitação urso
  - Controle de
    - Ex sando smartcards
  - Po gurança
    - eração periódica de senhas
  - Controle físico
  - ...

Esses mecanismos não conseguem mitigar todos os ataques

Como proteger o software? (2)

- Complementar com aplicação de técnicas de detecção de vulnerabilidades:
  - Análise de código
    - Manual (revisão, inspeção)
    - Automática (análise estática de código)
  - Realizar testes de segurança
    - Manual (hackers, tiger teams)
    - Automático (testes de penetração, injeção de ataques)

Análise estática de código

- Análise do código (fonte ou bytecode) em busca de:
  - Vulnerabilidades
  - Outras falhas
- É estática ➔ não executa o código
- Exemplos de ferramentas:
  - FindBugs
  - Yasca (Yet Another Source Code Analyzer)
  - …

Testes de penetração

- Testes realizados do ponto de vista do atacante:
  - Abordagem dinâmica, caixa preta
  - Uso de entradas maliciosas
    - Ex.: SQL Injection

```
public String auth (String login, String passw)
     trhow SQLException {

    String sql = "SELECT * FROM users WHERE "+
            "username=' " + login + " ' AND " +
             "password=' " + passw + " ' " ;

    "SELECT * FROM users WHERE username=' ' OR 1=1 -- ' AND
          password=' '";
}
```

Ferramentas para testes de penetração

- Também chamadas de *vulnerability scanners*
- Permitem automatizar os testes
- Existem inúmeras ferramentas:
  - Comerciais e open source
- Diferentes ferramentas buscam diferentes tipos de vulnerabilidades
- Exemplos de ferramentas:
  - HP WebInspect
  - IBM WatchFire
  - Rational AppScan
  - …
  - WSDigger
  - WSFuzzer
  - SoapUI Security Testing

Injeção de ataques

- Um tipo de teste de segurança
- Existem diversas técnicas, dentre as quais:
  - Emulação de ambientes hostis
  - Injeção de pacotes na rede
  - Alteração da sintaxe de mensagens
    - Envio de mensagens ou pacotes mal-formados
  - Emulação de ataques
    - Uso da Ballista
    - Uso de testes aleatórios (fuzzing)
    - Uso de modelo de ataques

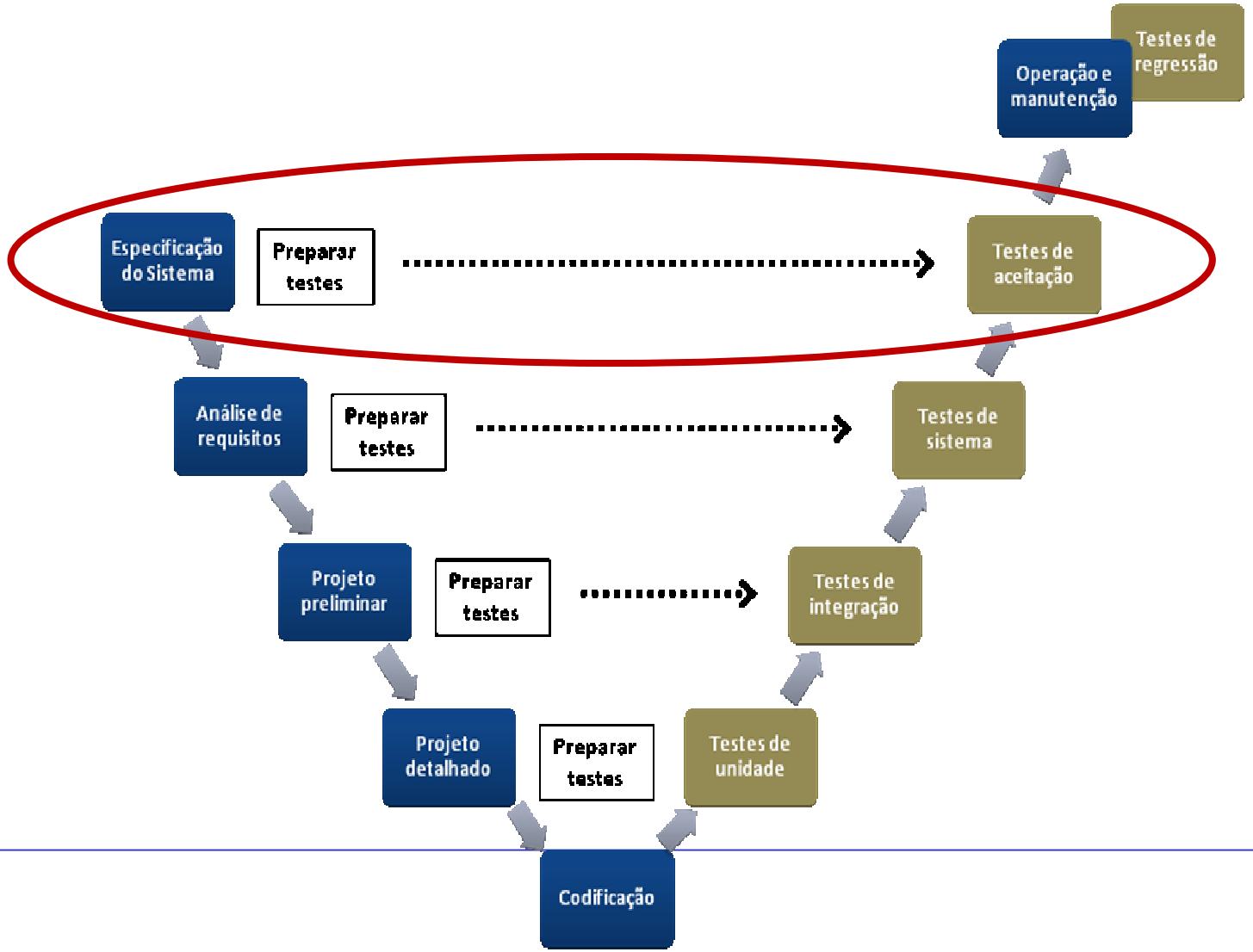
Recomendações para Testes de Segurança

- Critérios e metodologias para testes de segurança foram propostos por diferentes grupos:
  - NIST (National Institute of Standards and Technology)
    - Manual descrevendo técnicas a serem usadas nos testes de segurança
  - OSSTMM (Open Source Security Testing Methodology Manual)
    - desenvolvido pela ISECOM (Institute for Security and Open Methodologies)
    - Manual descreve a metodologia proposta para testes e análise de segurança
  - OWASP (Open Web Application Security Project)
    - Guia descrevendo melhores práticas para a realização de testes de penetração para aplicações e serviços Web http://www.owasp.org/index.php/Main_Page

Mais recomendações sobre testes de segurança

- Herzog, P. (2006). Open-source security testing methodology manual (OSSTMM 2.2). Technical report, Institute for Security and Open Methodologies (ISECOM).
- Wack, J., Tracy, M., and Souppaya, M. (2003). Computer security. Technical report, National Institute of Standards and Technology (NIST).

Testes no Processo de Desenvolvimento



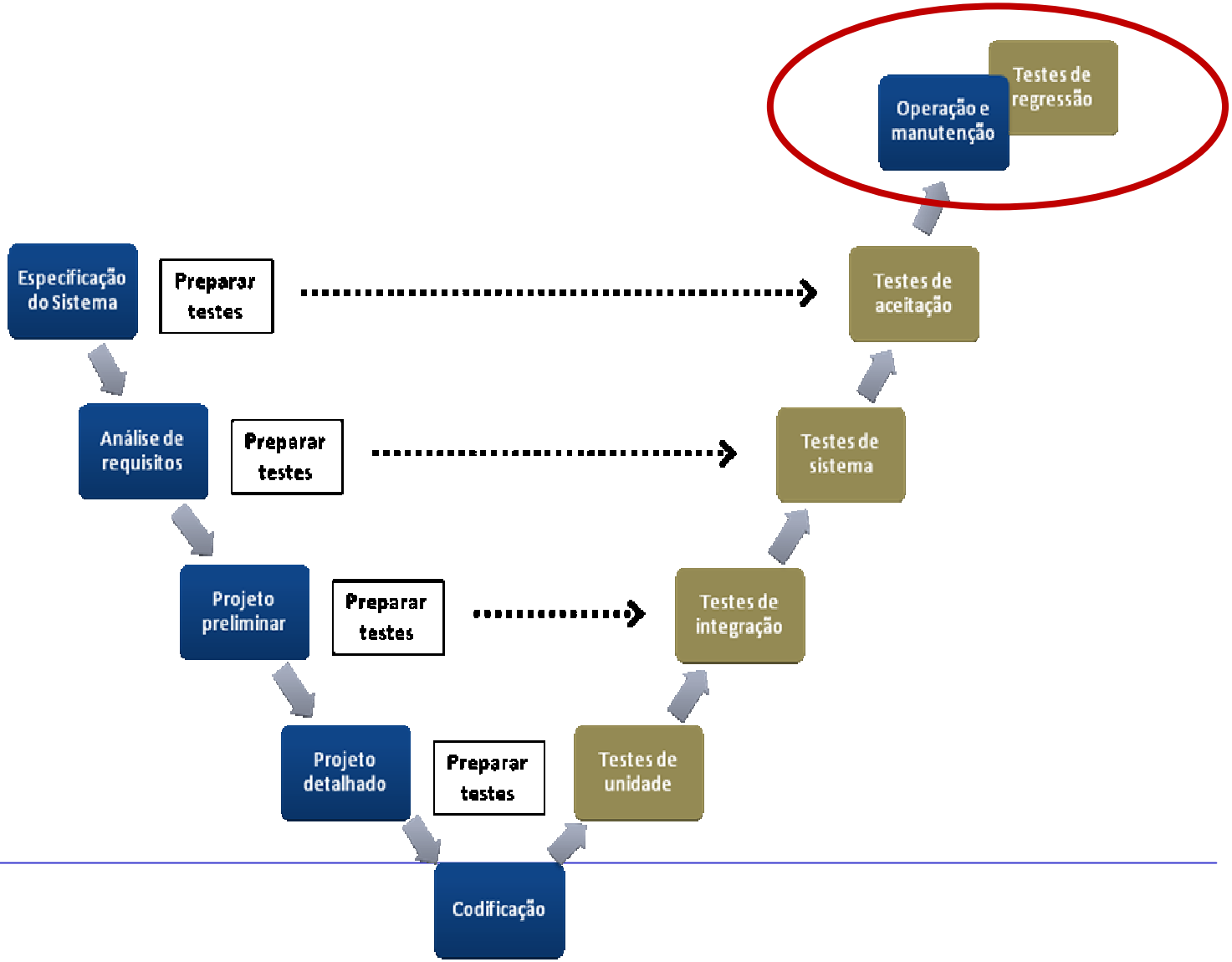
Testes de validação

- **Testes de aceitação**:
  - Realizados pelo cliente para decidir se aceita ou não o sistema
  - Realizados de acordo com um Plano de Aceitação
- **Testes de conformidade**:
  - Visam verificar se a implementação satisfaz à legislação ou a padrões estabelecidos
  - Podem ser realizados por centros de certificação

Testes de Aceitação

- Têm os mesmos objetivos que os testes de sistemas, só que envolvem a participação do cliente ou usuário
- Referências:
  - Manual do Usuário
- **Testes alfa**
  - Realizados por um grupo de usuários no ambiente de desenvolvimento
  - Visam determinar se o sistema pode ser liberado
- **Testes beta**
  - Realizados por um grupo de usuários em ambiente de operação

Testes no Processo de Desenvolvimento



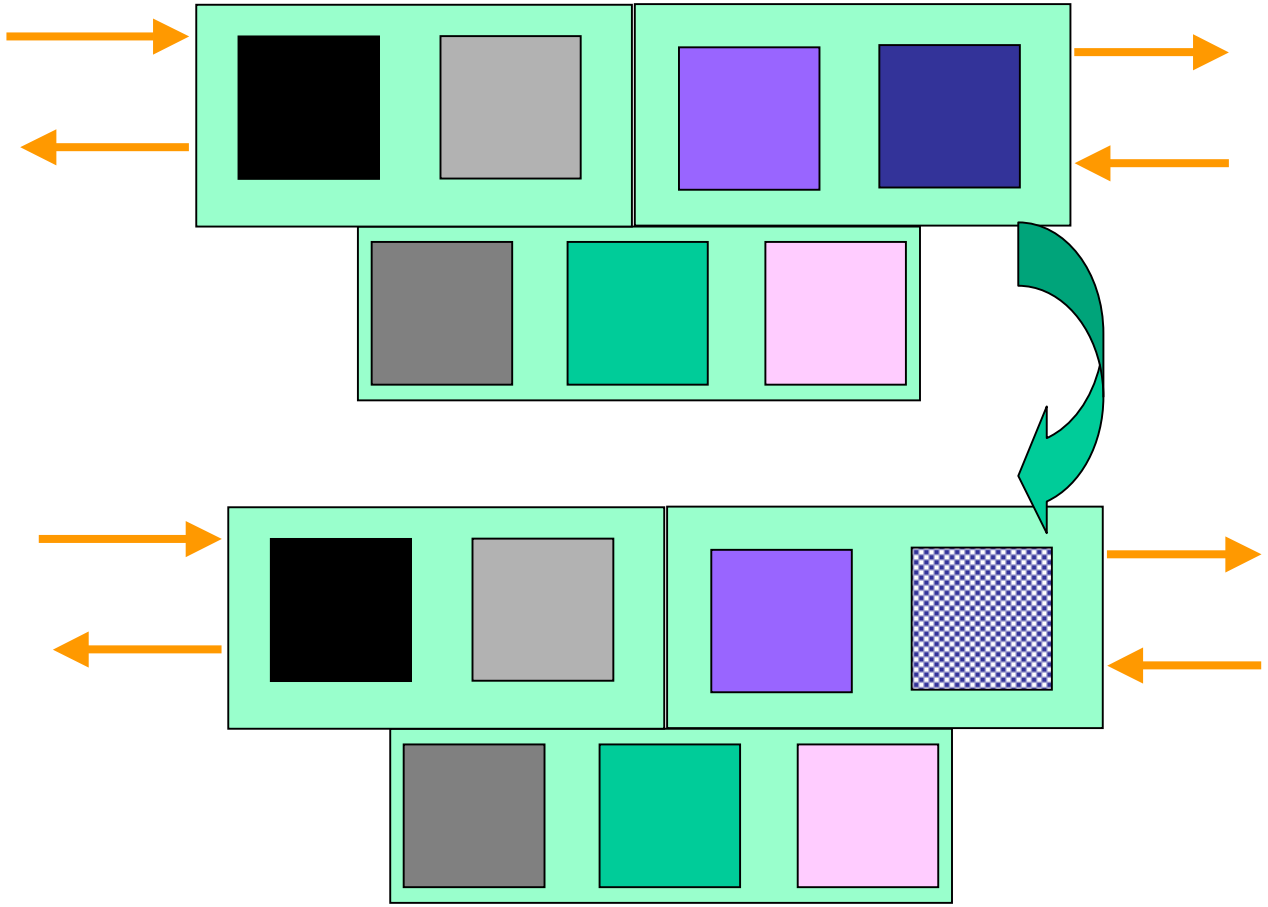
Testes de regressão

Testes realizados a cada vez que um sw é alterado

- Objetivo:
  - validar modificações feitas
  - mostrar que modificações realizadas não afetaram as partes que não foram modificadas

isto é

➲ mostrar que o sw não regrediu



Alguns conceitos

- Linha básica (“baseline”)
  - versão de um componente (ou sistema) já testada
- Delta
  - modificação feita a um componente (ou sistema) e que ainda não foi testada
- Configuração delta (“delta build”)
  - configuração executável do sistema contendo deltas e linhas básicas
- Caso de teste de regressão
  - caso de teste aplicado à linha de base com veredicto = passou
  - se veredicto = não passou na config. delta $\Rightarrow$ **falha de regressão**

Quando aplicar

- Para testar aplicações críticas que devem ser retestadas freqüentemente
- Para testar sw que é alterado constantemente durante o desenvolvimento (ex.: Processo Incremental ou Evolutivo)
- Para testar componentes reutilizáveis para determinar se são adequados para o novo sistema
- Durante os testes de integração
- Durante os testes, após correções
- Em fase de manutenção (corretiva, adaptativa, perfectiva ou preventiva)
- Para identificar diferenças no comportamento do sistema quando há mudanças de plataforma (uso de seqüências-padrão ou “benchmarks”)

Quando aplicar - OO

- Quando uma nova subclasse é criada
- Quando uma super-classe é alterada
- Quando uma classe servidora é alterada
- Quando uma classe é reutilizada em um novo contexto

Abordagens

- Dados:
  - P: Programa original — P' – programa modificado (delta)
  - T: conjunto de testes existente — T' – conjunto de testes de regressão
- Como selecionar T'?

- Retesta tudo: T' = T
- Seletiva: T'$\subset$ T

Considerações sobre a seleção

- Problema: segurança (*safety*)
  - como obter T' contendo casos de teste t ∈ T que exercitem código de P que foi modificado em P'?

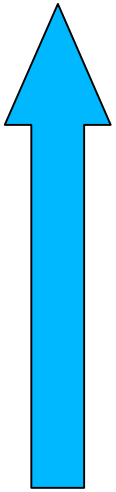
    ☹ Problema indecidível
- O uso de uma seqüência de regressão segura

  ⇒ todos os casos de teste que podem revelar a presença de falhas foram aplicados

  ⇏ ausência de falhas de regressão ou de qualquer outro tipo de falha

Técnicas

- As técnicas de seleção de testes de regressão podem ser baseadas:
  - No código
    - Grafo de fluxo de controle
  - Na arquitetura
    - firewall
  - Na especificação
    - Casos de uso



Seleção baseada no código

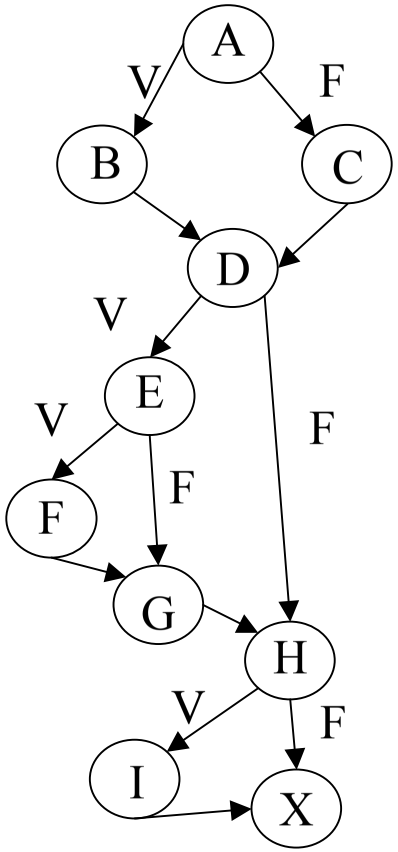
- Exemplo de técnica:
  - Seleção baseada em segmento modificado
- As técnicas se baseiam na construção do Grafo de Fluxo de Controle (GFC) do programa
  - Passeio síncrono no grafo original e no grafo modificado para identificar as modificações

Exemplo: P e T

P

```
if A then B
else C;
if D then
   if E then F;
   G;
if H then I;
X;
```

G



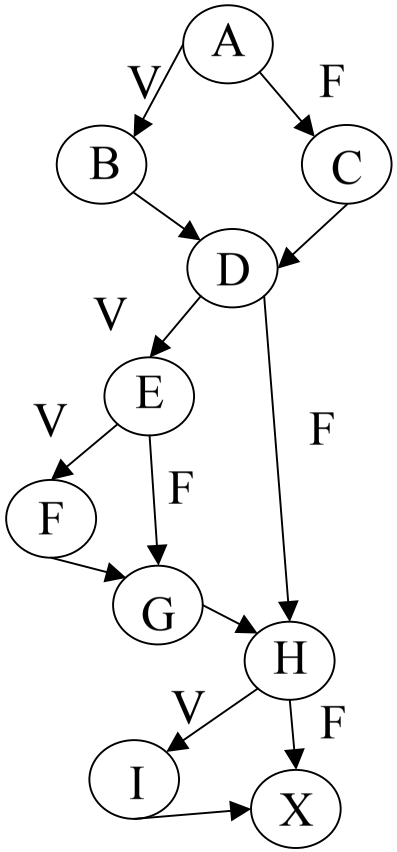
T

testes	caminho
t1	ABDEFGHIX
t2	ABDEGHIX
t3	ABDHIX
t4	ACDHX

Exemplo: P'

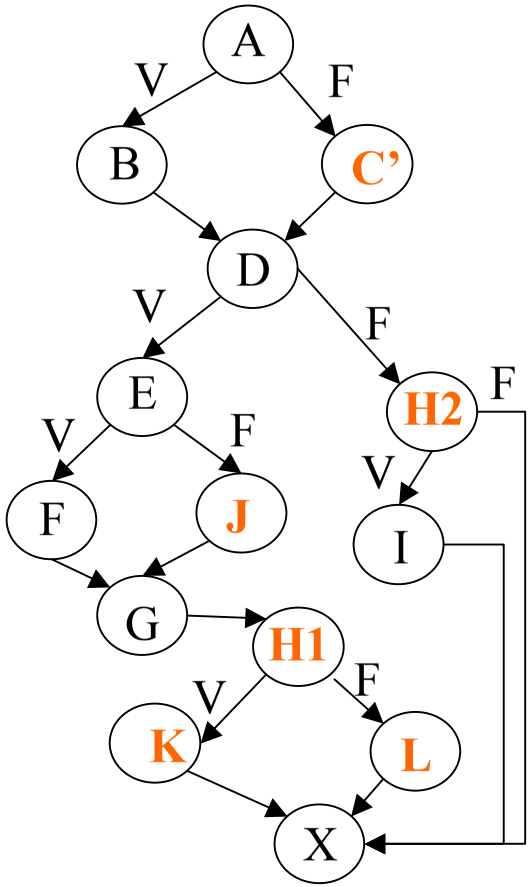
P
```
if A then B
else C;
if D then
   if E then F;
   G;
if H then I;
X;
```

G



P'
```
if A then B
else C';
if D then
   if E then F
   else J;
   G;
   if H1 then K
   else L;
else if H2 then I;
X;
```

G'



Exemplo – seleção de T’

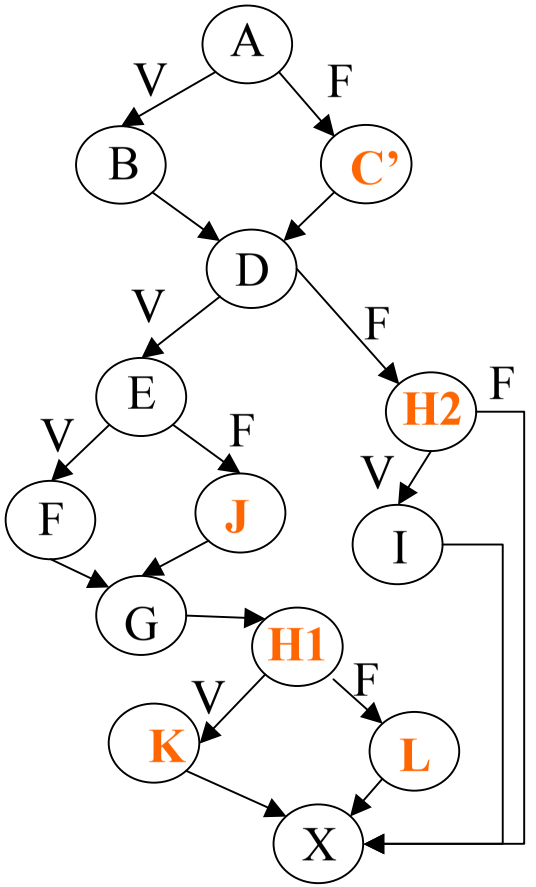
T’

Modificação	seq. segura mínima
C → C’	t4 (**AC**DHX)
+ J	t2 (ABD**E**G**H**IX)
H → H1 + H2	t1 (ABDEF**GH**IX),
	t2 (ABDE**GH**IX),
	t3 (AB**DH**X),
	t4 (AC**DH**X)

P’

```
if A then B
else C’;
if D then
   if E then F
   else J;
   G;
   if H1 then K
   else L;
else if H2 then I;
X;
```

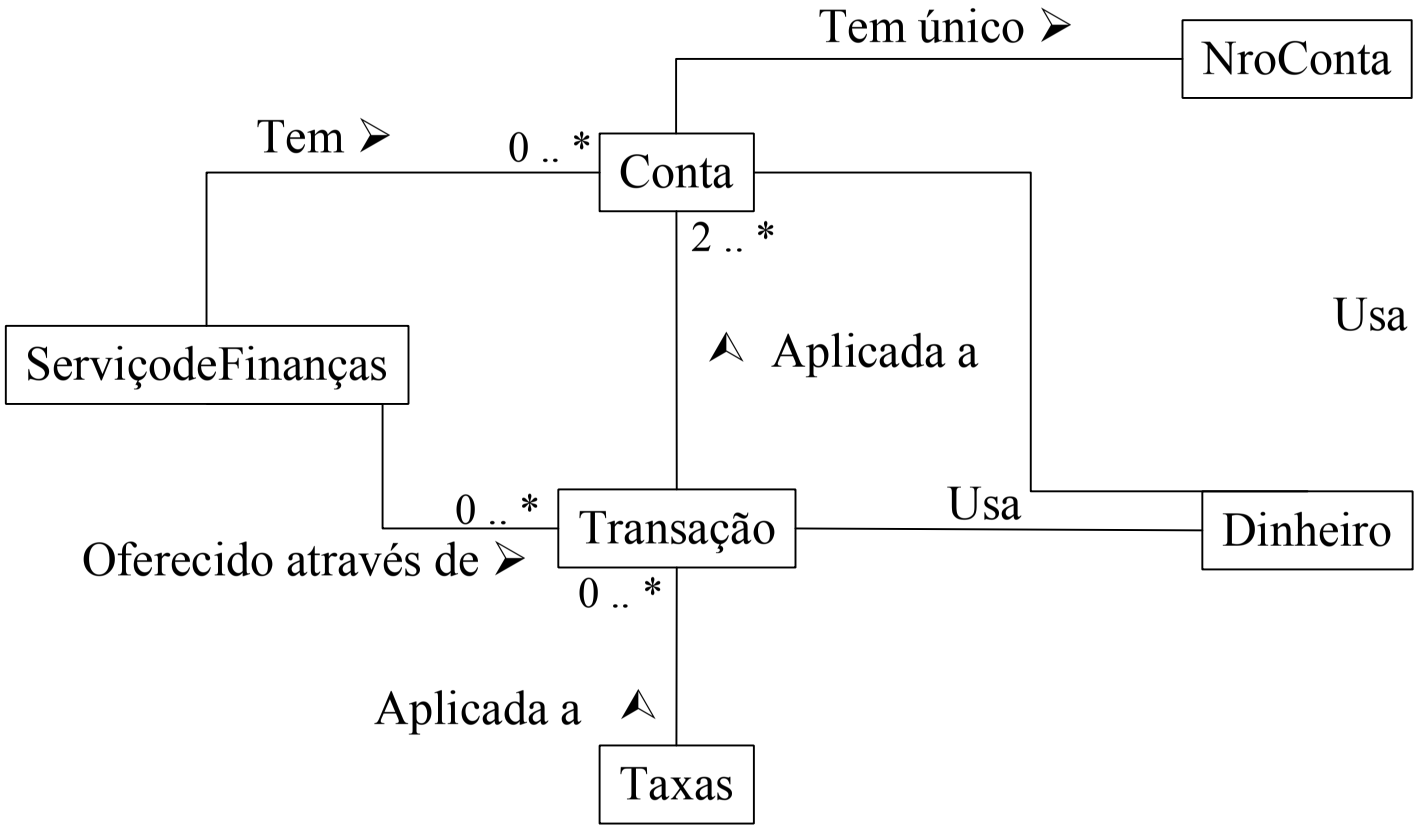
G’



Seleção baseada na arquitetura

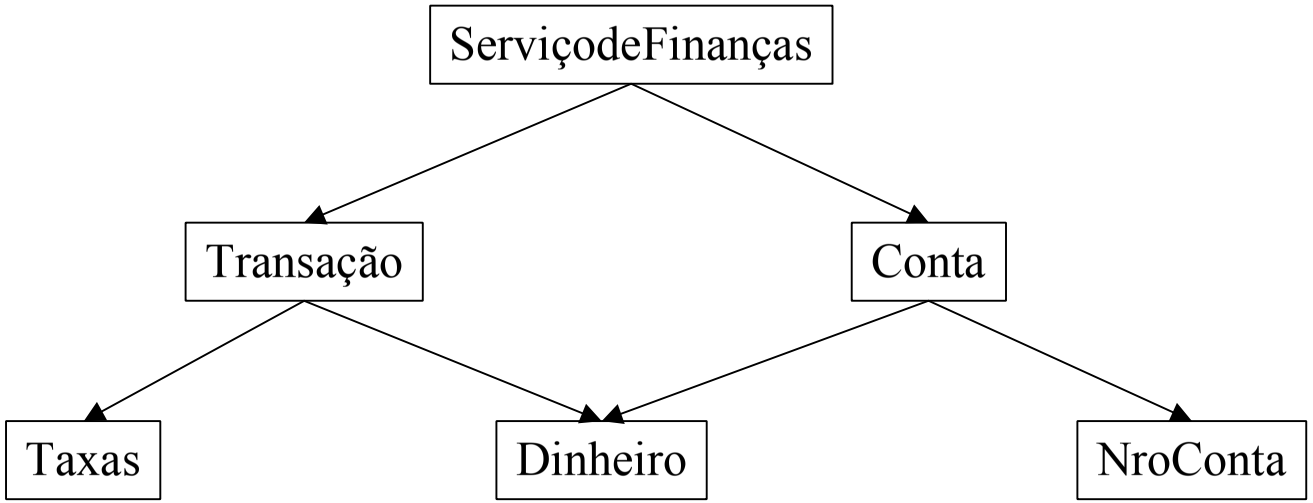
- Uso de *firewall*
  - O conceito de firewall foi introduzido por Leung e White (1989) ***para separar os módulos que podem ser afetados pelas modificações dos outros.***
  - Uma vez identificado o firewall,é selecionado um subconjunto de testes que ***exercitem os módulos dentro do firewall.***
  - A determinação do firewall se dá através da ***análise de dependências:***
    - B usa A
    - B é servidor de A
    - B é subclasse de A
    - …

Exemplo – modelo de classes



Exemplo – diagrama de dependências

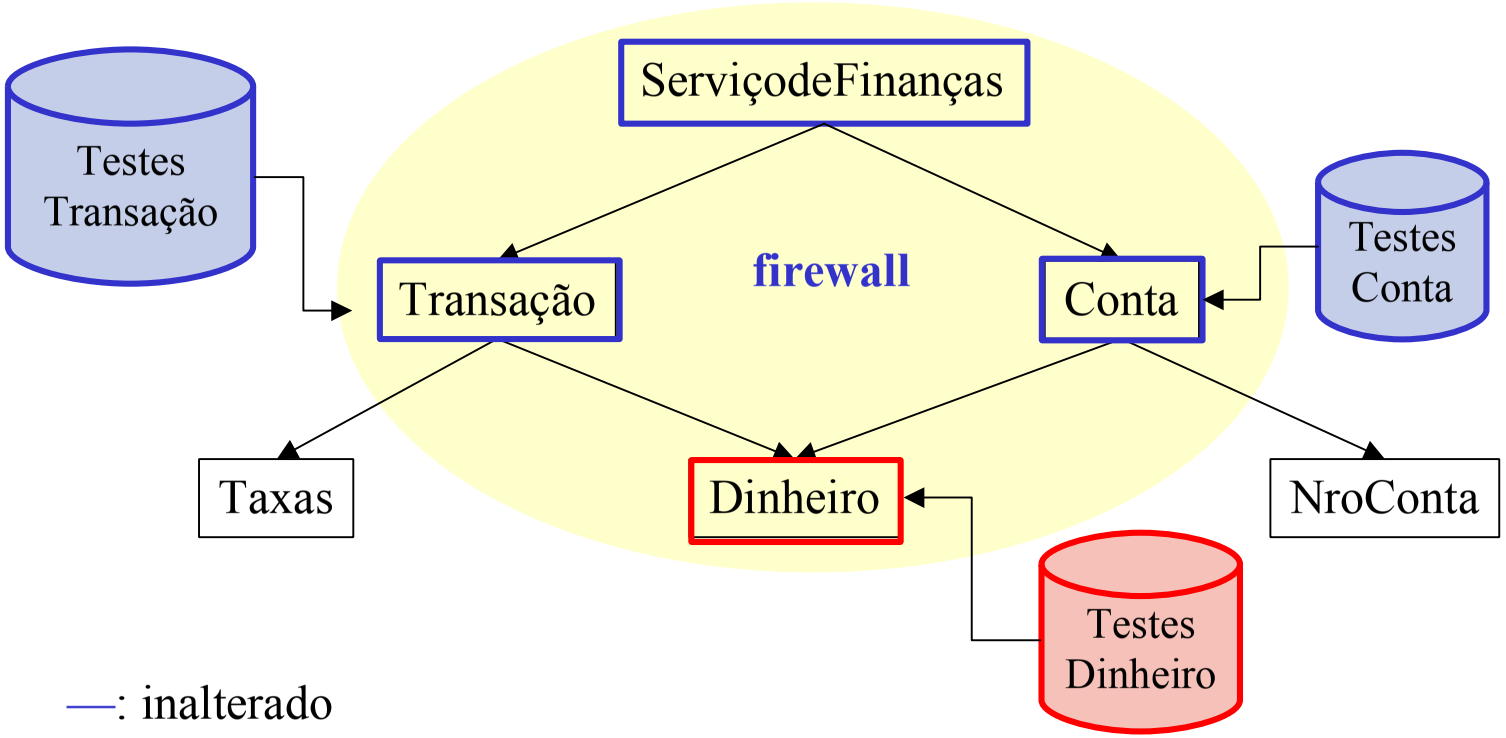
Dependências entre os componentes



→: depende de

Exemplo – obtenção do *firewall*

Seleção de Testes no Firewall



—: inalterado
—: modificado

Técnicas baseadas na especificação

- A seleção baseia-se na análise do modelo de especificação.
- O mais comum: casos de uso [Binder99, c.15] :
  - Casos de uso de maior risco
  - Casos de uso mais freqüentes
- Técnicas baseadas nos casos de uso podem usar a matriz de rastreabilidade para seleção dos testes

Matriz de rastreabilidade

	t_1	t_2	...	t_M
Caso de uso 1	✓	✓		
Caso de uso 2		✓		
...				
Caso de uso N		✓		✓

Principais pontos aprendidos