2 Contexto de Teste de Software

A satisfação de um usuário de software está diretamente relacionada ao fato deste software atender aos seus requisitos funcionais e não funcionais sem apresentar falhas durante a sua execução. No entanto, apesar dos esforços dos desenvolvedores, erros são cometidos ao longo de todo o ciclo de desenvolvimento de um software.

Uma especificação mal concebida pode originar um sistema com funcionalidades distintas das esperadas pelo usuário ou, até mesmo, ausentes. Uma análise mal elaborada pode resultar numa arquitetura que não permita atender determinados critérios de qualidade. Um projeto mal construído pode comprometer a evolução do software. Uma codificação mal feita pode ocasionar falhas de execução.

Defeitos são introduzidos durante a codificação do software e acabam se manifestando sob a forma de falhas muitas vezes observadas somente pelo usuário. Algumas causas podem ser apontadas como precursoras desta situação (Whittaker 2000). O usuário pode ter executado um código não testado. A ordem em que os comandos são executados durante o uso do software difere dos cenários de teste. O usuário aplicou uma combinação de valores de entrada não testada. O ambiente operacional do usuário nunca foi testado.

Estes problemas revelam a necessidade de ***disciplina na condução de testes de um software a fim de identificar, tão cedo quanto possível, os defeitos nele existentes***. Como observado, planejar e executar testes de software requer atenção sobre vários aspectos que incluem as funcionalidades, as entradas, as combinações de entradas e o ambiente de execução do software.

Este capítulo aborda o contexto de teste de software no qual se situa esta pesquisa. Define a terminologia de teste adotada (seção 2.1). Introduz um processo de teste (seção 2.2) comum a esta prática. Cita as diferenças de escopo e de tipo de teste (seção 2.3), destacando o tipo de teste funcional (seção 2.4). Trata da automação de teste (seção 2.5), do teste baseado em modelo (seção 2.6) e de

PUC-Rio - Certificação Digital Nº 0611931/CA

sua extensão, o teste dirigido por modelo (seção 2.7). Ao final, revela os desafios projetados para a área de teste (seção 2.8).

2.1. Terminologia de Teste

A terminologia de teste aplicada na descrição desta pesquisa está em conformidade com especificações e padrões reconhecidos e adotados pela academia e pela indústria. O alinhamento entre os conceitos empregados pela prática de teste e estas referências não só facilita a sua compreensão quanto garante a sua adequação a contextos e domínios previamente conhecidos.

As referências utilizadas foram o glossário padrão de terminologia de engenharia de software (IEEE Std 610.12, 1990), o vocabulário usado em teste de software (BS 7925-1, 1998) e a segunda versão do UML (*Unified Modeling Language*) *Testing Profile* (UML2TP, 2003).

Os dois glossários são padrões internacionais e foram elaborados, respectivamente, pelo *Institute of Electrical And Electronics Engineers* e pela *British Standards Institution*. O UML *Testing Profile* é uma especificação do OMG (*Object Management Group*) que define uma linguagem para projeto, visualização, especificação, análise, construção e documentação de artefatos de sistema de teste. Esta linguagem pode ser usada com as principais tecnologias de objetos ou componentes e com sistemas de teste em diversos domínios de aplicação.

Os principais termos e expressões serão descritos brevemente segundo estas especificações de teste de software.

O ***sistema em teste*** (*system under test* ou SUT) é a parte ou todo sistema, subsistema ou componente em teste. Funciona como uma caixa-preta que somente pode ser exercitada pela estrutura de teste por meio das operações disponíveis em sua interface pública (UML2TP, 2003).

O ***cenário de teste*** (*test scenario ou test context*) é a coleção de casos de teste reunida com a configuração de teste. Os casos de teste são executados com base no cenário de teste (UML2TP, 2003).

PUC-Rio - Certificação Digital Nº 0611931/CA

O ***critério de teste*** (*test criteria*) representa o critério que o sistema tem de atender para passar no teste. O critério de aceitação (*acceptance criteria*) de um usuário ou interessado (*stakeholder*) e as regras de decisão de que um sistema passou ou falhou nos testes (*pass/fail criteria*) são exemplos de critério de teste (IEEE Std 610.12, 1990).

O ***ambiente de teste*** (*test environment* ou *test configuration*) é a descrição do ambiente de hardware e software no qual os testes são executados e a descrição de qualquer software com que o sistema em teste interage quando testado usando dispositivos de teste ou *stubs, que são* esqueletos ou implementações com propósito especial de um módulo de software (BS 7925-1, 1998).

O ***conjunto de teste*** (*test case suite* ou *test suite*) é a coleção de um ou mais casos de teste de um sistema em teste (BS 7925-1, 1998).

O ***caso de teste*** (*test case*) é o conjunto de dados de entrada, condições de execução e resultados esperados elaborados para atingir um objetivo específico de teste, tal como exercitar um fluxo do programa ou verificar a conformidade com um requisito específico (IEEE Std 610.12, 1990; BS 7925-1, 1998).

O ***objetivo de teste*** (*test goal* ou *test objective*) é o conjunto de características de um software a serem medidas em condições específicas pela comparação do comportamento atual com o comportamento requerido descrito na documentação do software (IEEE Std 610.12, 1990).

O ***dado de teste*** (*test data* ou *stimulus*) é o dado enviado ao sistema em teste (UML2TP, 2003). Corresponde freqüentemente ao dado de entrada fornecido à interface pública do sistema em teste durante a execução do caso de teste.

O ***oráculo de teste*** (*test oracle*) é o mecanismo usado para gerar o resultado esperado de um caso de teste. Este resultado é confrontado com o resultado real obtido pela execução do sistema em teste. (BS 7925-1, 1998).

O ***resultado observado*** (*test result* ou *observation*) é o dado que reflete a reação do sistema em teste a um dado de teste (UML2TP, 2003). Corresponde comumente ao dado de saída (ou retorno) de uma função em teste presente na interface pública de um sistema.

O ***laudo de teste*** (*test log*) é o registro da interação resultante da execução de um caso de teste. O laudo está associado a um veredicto que indica o grau de conformidade do sistema em teste com o objetivo definido por um caso de teste (UML2TP, 2003).

PUC-Rio - Certificação Digital Nº 0611931/CA

O ***veredicto*** (*verdict*) é a sentença conferida ao sistema em teste indicando a sua corretude ou não. Pode ser empregado também para reportar falhas no sistema de teste. Seus possíveis valores são "passou" (*pass*), "falhou" (*fail*), "inconclusivo" (*inconclusive*) ou "erro" (*error*). Os dois primeiros valores estão relacionados aos resultados do teste que pode ter seu propósito, respectivamente, evidenciado ou invalidado. O valor inconclusivo só é atribuído quando o teste nem passa nem falha. Na prática, isso só acontece se o teste elaborado não permite extrair conclusões sobre o resultado. O último indica uma falha na própria estrutura de teste que a torna, por conseguinte, inconsistente ou inválida (UML2TP, 2003).

2.2. Processo de Teste

PUC-Rio - Certificação Digital Nº 0611931/CA

Pautado pelas considerações em relação ao planejamento e à execução das atividades de teste de software, este pode ser estruturado em fases seqüenciais (Figura 3). Nesta abordagem (Whittaker, 2000), cada fase trata de problemas relacionados e o resultado de seu processamento serve como base para a fase seguinte.

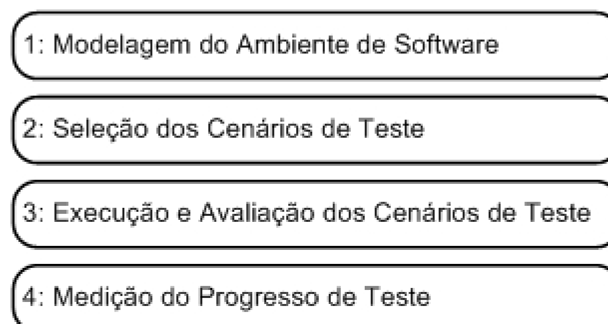

Figura 3 – Processo de Teste

O primeiro passo é a ***modelagem do ambiente de software*** (Figura 3). Nesta fase, é preciso identificar e simular as interfaces que o sistema de software utiliza bem como enumerar as possíveis entradas de dados aplicadas a estas interfaces. O termo interface aqui empregado representa não só a interface entre

humano e computador, mas também as interfaces de software, de sistemas de arquivos e de comunicação. A escolha do escopo de teste tem implicações diretas sobre os custos e o tempo gasto para a conclusão da atividade de teste. Além disso, é necessário compreender também as interações do usuário que estão fora do escopo do sistema em teste e podem afetar a sua operação (ex. leitura de um arquivo que pode ser apagado por outro usuário). Cada ambiente de aplicação pode resultar num número significativo de interações a testar. Um modelo é utilizado para descrever estas interações.

O segundo passo é a ***seleção dos cenários de teste*** (Figura 3). Os parâmetros normalmente empregados na definição dos casos de teste são a cobertura do código e a cobertura do domínio dos dados de entrada. Mas essas coberturas analisadas de forma isolada não são suficientes. A seqüência de execução de comandos de código, a seqüência de aplicação de entradas e o ambiente de teste (configurações de hardware e software) também influenciam na formação dos cenários de teste. Como o número de cenários de teste para garantir estes aspectos pode ser infinito, é necessário estabelecer critérios que permitam definir a completeza dos testes de modo a atender os objetivos claramente determinados para os testes segundo um cronograma realista e factível. Estes critérios são limitações impostas à metodologia de teste que afetam os resultados reportados.

O terceiro passo corresponde à ***execução e avaliação dos cenários de teste*** (Figura 3). A etapa inicial desta fase consiste em converter os cenários de teste numa forma executável que simula as ações dos usuários. A aplicação dos cenários pode ser manual ou automatizada. A primeira demanda muito esforço e está mais sujeita a erros humanos, a segunda requer a simulação de cada dado de entrada e dado de saída de todo ambiente operacional. A etapa final envolve comparar o resultado obtido pela execução do cenário de teste com o resultado esperado documentado pela especificação do software. Esta avaliação pode ser feita por meio de formalização da especificação, que permite derivar design e código usados para comparar comportamentos esperados e observados, ou por intermédio de código de teste incluído no código do software.

O último passo equivale à capacidade de ***medição do progresso de teste*** (Figura 3). Considerando dados de entrada, cobertura de código, execuções com sucesso, falhas encontradas e outras estatísticas, observa-se que a simples

PUC-Rio - Certificação Digital Nº 0611931/CA

contagem de números pode não representar o progresso do teste corretamente. Qual seria o significado de encontrar muitas falhas? Uma boa metodologia de teste ou um código extremamente defeituoso? Deste modo, identificar o final de um teste de software é uma tarefa muito difícil, visto que sempre se desconhecerá o número real de defeitos remanescentes. Por isso, é preciso estabelecer critérios de término dos testes fundamentados em graus de complexidade de teste (completeza estrutural) e confiabilidade (completeza funcional) acordados como satisfatórios.

2.3. Escopo e Seleção de Teste

O teste de software é classificado (Whittaker, 2000) de acordo com a primeira e a segunda fase do processo de teste descrito.

Segundo o escopo da modelagem do ambiente de software, o teste pode ser um teste de unidade, teste de integração ou teste de sistema. O ***teste de unidade*** avalia individualmente um componente de software ou uma pequena coleção de componentes. O ***teste de integração*** procura analisar a comunicação entre os componentes por meio de suas interfaces. O ***teste de sistema*** considera a coleção de todos os componentes que compõem um software, sendo todo o domínio normalmente avaliado em relação à sua especificação. Este teste não deve ser confundido com o ***teste de aceitação*** ou auditoria funcional que visa verificar se o sistema atende às necessidades do usuário e não apresenta problemas ao ser utilizado.

Quando o quesito é a seleção de caso de teste, existem o teste estrutural, o teste da estrutura de dados e o teste funcional. No ***teste estrutural***, os cenários de teste consideram somente a estrutura do código e as estruturas de dados existentes. É conhecido também como ***teste caixa-branca*** ou ***teste baseado no código***. No ***teste da estrutura de dados***, a cobertura é medida com relação a uma parte da estrutura do código em adição à especificação. É conhecido ainda como teste ***caixa-entreaberta*** ou ***teste caixa-cinza***. O ***teste funcional***, ao contrário do estrutural, não é influenciado por estas estruturas. Este tipo de teste será abordado em mais detalhes a seguir por ser o empregado neste trabalho.

PUC-Rio - Certificação Digital Nº 0611931/CA

2.4. Teste Funcional

No ***teste funcional***, a seleção dos cenários de teste se baseia nas características reveladas pela especificação do sistema em teste ou pelo ambiente operacional. A estrutura do código fonte ou as estruturas de dados internas (não expostas) não servem como referencial para elaboração dos testes.

A idéia principal de um teste funcional é ignorar os mecanismos internos de um sistema ou de um componente e focar somente nas saídas geradas em resposta às entradas selecionadas e às condições de execução do software (Gao et al., 2003). Desta forma, mesmo nos casos em que o código fonte não esteja disponível, é possível testar um software e verificar se ele atende aos requisitos do usuário.

Como não depende do código, a adequação do teste funcional está atrelada a outros fatores. A existência de uma especificação do sistema em teste é crucial para que requisitos funcionais sejam identificados. A capacidade do software em teste ser configurado de acordo com propriedades que podem ser redefinidas altera o seu comportamento em relação ao ambiente operacional. As interfaces do sistema são a única forma de interação com o mesmo e, portanto, precisam ser bem compreendidas para que a eficiência e a eficácia do teste sejam obtidas.

O teste funcional é conhecido também como ***teste caixa-preta***, ***teste baseado na especificação*** ou ***teste comportamental***.

PUC-Rio - Certificação Digital Nº 0611931/CA

2.4.1. Técnicas de Teste Funcional

As interfaces de cada componente do software representam a forma de interação com o mesmo e, por conseguinte, as diferentes técnicas de teste funcional (Gao et al., 2003) fazem uso da especificação destas interfaces para elaborar os cenários de teste. Algumas destas técnicas são descritas brevemente a seguir.

O ***teste randômico*** (Figura 4) adota a estratégia da seleção aleatória dos casos de teste de todo o domínio de entrada. Esta abordagem não garante uma confiabilidade completa em relação ao teste das entradas por trabalhar com amostras que podem não ser significativas para os propósitos do teste.

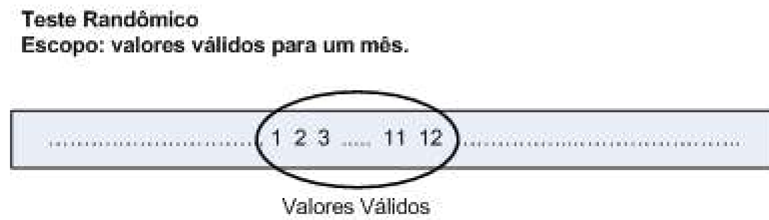

Figura 4 – Exemplo de Teste Randômico

O ***teste de partições equivalentes*** (Figura 5) procura superar a limitação do teste randômico dividindo o domínio de entrada em diferentes partições disjuntas. A propriedade principal desta técnica é pressupor-se que, se um elemento de uma partição provocar a falha de um teste, todos os demais membros da partição também provocarão falhas. O contrário também é verdade, se um elemento de entrada resulta num teste bem sucedido, então os outros elementos da mesma partição também o serão. Este comportamento permite avaliar o domínio inteiro sem redundância, bastando avaliar poucos elementos de cada partição. Contudo, a maior dificuldade está em se definir de forma sistemática as partições.

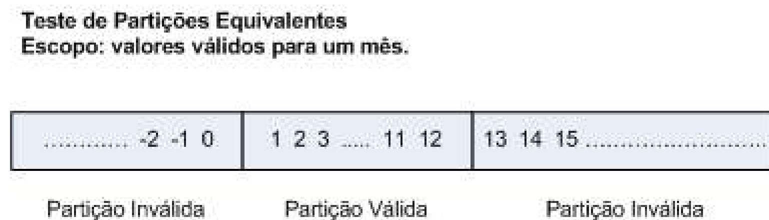

Figura 5 – Exemplo de Teste de Partições Equivalentes

O ***teste de valores de contorno*** (Figura 6) é uma extensão do teste de partições equivalentes. O princípio que rege esta técnica é o de que cada elemento de uma partição não apresenta a mesma probabilidade de revelar ou esconder um defeito do sistema em teste. Pela experiência adquirida na programação de aplicações, percebe-se que os elementos próximos ao contorno das partições normalmente expõem falhas com mais freqüência. Assim, enquanto o teste de partições equivalentes aumenta a eficiência do testes por reduzir o número de casos de teste, o teste de valores de contorno o suplementa focando nas áreas de maior risco.

PUC-Rio - Certificação Digital Nº 0611931/CA

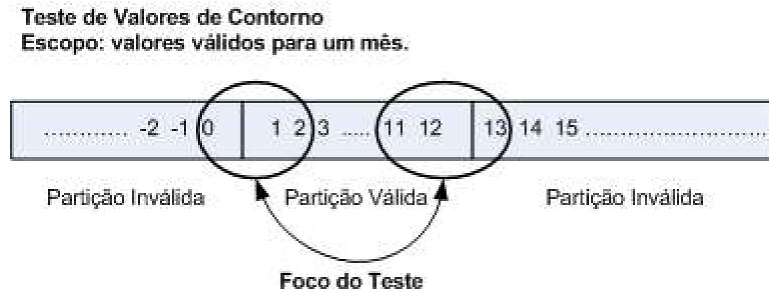

Figura 6 – Exemplo de Teste de Valores de Contorno

O ***teste baseado em tabelas de decisão*** (Figura 7) é aplicado nos casos em que não é possível assumir variáveis mutuamente independentes na análise como nos teste de partições equivalente e teste de valores de contorno. Este técnica aborda cenários mais complexos onde há uma relação de dependência entre as variáveis e onde a combinação de ações é influenciada pela lógica dos relacionamentos entre as variáveis. As tabelas de decisão refletem todas as condições de teste, suas diferentes combinações, as ações relacionadas e suas correspondentes respostas.

Teste baseado em Tabela de Decisão
Escopo: valores válidos para classificação de aluno de pós-graduação.

Condições	Aluno de Mestrado	Sim	Sim	Não	Não
	Profissional da Indústria	Sim	Não	Sim	Não
Ações	Tempo Parcial	X	X		
	Tempo Integral		X		

Figura 7 – Exemplo de Teste baseado em Tabelas de Decisão

O ***teste de mutação*** (Figura 8) é um teste baseado em defeitos. O mutante é uma versão do sistema em teste conhecida e deliberadamente defeituosa. Como o código fonte não está disponível, as alterações no programa alternativo são promovidas no nível de interface. O resultado de sua execução é então comparado com a do programa original. Caso sejam equivalentes, o mutante não contribui para a análise. Do contrário, se o mutante for detectado, significa que a alteração foi identificada pelo conjunto de teste e maior é adequação do mesmo.

PUC-Rio - Certificação Digital Nº 0611931/CA

Usam-se mutantes para aferir a sensibilidade do teste. Se o teste deixa de observar um defeito inserido num mutante, ele possivelmente deixará de encontrar defeitos originais existentes. O teste de mutação mede a eficácia do teste. Se o teste for altamente eficaz é de se esperar que o número de defeitos residuais seja bem pequeno.

Teste de Mutação
Escopo: valores válidos para classificação de aluno de pós-graduação.

Código Original	Código Mutante
SE (Aluno-Mestrado **E** Profissional) ENTÃO Resultado = Tempo-Parcial;	SE (Aluno-Mestrado **OU** Profissional) ENTÃO Resultado = Tempo-Parcial;

Nota: no código mutante, se o aluno de pós graduação não for aluno de Mestrado e for um profissional da indústria, então será classificado como aluno de tempo parcial, que não é o valor esperado para esta situação.

Figura 8 – Exemplo de Teste de Mutação

2.4.2. Características de uma Ferramenta de Teste Funcional

Independente das técnicas de teste funcional adotadas, uma ferramenta de teste funcional deve apresentar algumas características que a tornem útil e amplamente empregada ao longo do ciclo de desenvolvimento de um software. Segundo estudo recente (Andrea, 2007), tais características quando existentes podem ajudar a garantir o sucesso inclusive do programa de testes.

A primeira delas consiste em prover um ambiente em que seja possível descrever facilmente os cenários de teste. Se a definição destes cenários for complicada, isto representará um gargalo para a adoção da ferramenta que logo será abandonada. Idealmente, a ferramenta deve empregar uma linguagem de teste bem definida que permita especificar requisitos de forma clara e efetiva. Isto, somado a recursos do ambiente de desenvolvimento hoje existentes para codificação de sistemas (ex. preenchimento automático de comandos – "*code complete*"), facilitará a composição e a manutenção dos testes, diminuindo o impacto de sua adoção, que pode torná-los incompletos e obsoletos.

PUC-Rio - Certificação Digital Nº 0611931/CA

Outro aspecto relevante é permitir uma fácil leitura e compreensão dos cenários de testes. Até mesmo uma pessoa não especializada em teste terá condições de acompanhar a verificação de completeza e corretude do mesmo. Portanto, precisa ser concebida de forma a atender diferentes papéis no desenvolvimento de um produto. Logo, contempla o usuário, o especialista em negócios, o desenvolvedor e o testador. Uma vez garantido um formato legível, este objetivo será alcançado.

A ferramenta também deve comportar a manutenção dos cenários de teste tornando possível a sua re-execução e a sua modificação. A re-execução possibilita a reprodução da falha, desde que preservado o estado anterior do cenário de teste. Também permite a realização de ***teste de regressão***, por meio do qual é possível avaliar se determinados cenários de teste continuam válidos após mudanças introduzidas no código do sistema em teste.

Para tornar seu emprego mais amplo, o ideal é que funcione em diferentes ambientes de execução de acordo com a necessidade do usuário de teste. O resultado da execução da ferramenta precisa ser uma saída com significado que facilitará a validação de oráculos de teste.

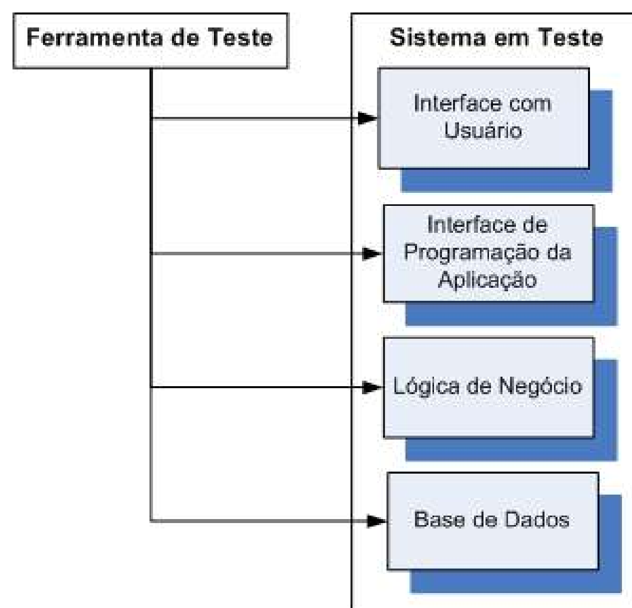

Figura 9 – Interação: Ferramenta e Sistema em Teste (Adaptação: Andrea, 2007)

Visando contribuir com a automação dos testes, a ferramenta pode ainda apresentar diferentes pontos de contato (interfaces) com o sistema em teste (Figura 9). A cada ponto de contato estará associada uma camada da aplicação.

PUC-Rio - Certificação Digital Nº 0611931/CA

Assim é possível executar testes de forma mais rápida nos casos em que a interação evita camadas não pertinentes ao cenário de teste.

Embora seja óbvio, é necessário ressaltar que a corretude de operação da ferramenta de teste é um caráter fundamental. Ela não pode mascarar com erros próprios os erros existentes no sistema em teste.

2.5. Automação do Teste

Em virtude do teste manual de software representar uma atividade tediosa, muitas vezes redundante, sujeita a falhas humanas e um desperdício de tempo e dinheiro, a automação desta prática passou a ser vista como uma alternativa atraente para a aplicação de testes no desenvolvimento do software.

A ***automação de teste de software*** refere-se às atividades de automação das tarefas e operações de engenharia no processo de teste de software utilizando estratégias bem definidas e soluções sistemáticas (Gao et al., 2003). Com isso, procura acelerar o processo de teste, reduzindo seu custo e tempo ao longo do ciclo de vida do software. Aumenta a qualidade e a eficácia do processo de teste ao permitir a adequação de um critério de teste num cronograma bem definido.

2.5.1. Níveis de Automação de Teste

O grau desta automação varia de ambiente para ambiente de desenvolvimento. Com o intuito de avaliar o nível de maturidade da automação de teste de software no processo de teste adotado, um estudo (Gao et al., 2003) propôs a seguinte classificação:

- ***Nível 0 – Sem ferramenta de teste.***
  Estágio em que o processo de teste é manual e não é auxiliado por ferramentas em nenhuma de suas fases.
- ***Nível 1 – Inicial.***
  Estágio em que o processo de teste provê somente soluções sistemáticas para a gestão de informações de teste, incluindo requisitos de teste, casos de teste, procedimentos de teste, resultados dos testes e relatórios de falhas.

PUC-Rio - Certificação Digital Nº 0611931/CA

- ***Nível 2 – Repetível.***

  Estágio que provê os recursos do nível anterior e acrescenta ao processo de teste a capacidade de execução dos testes e validação de resultados de forma sistemática. Ferramentas de análise de cobertura de teste baseado em código podem ser usadas neste nível.

- ***Nível 3 – Automático.***

  Estágio que provê os recursos do nível repetível e conta com a presença de geradores sistemáticos de casos de teste. Os dados e oráculos de teste são gerados automaticamente. Todo o processo de teste é automatizado.

- ***Nível 4 – Otimizado.***

  Estágio em que o processo automatizado conta com soluções de medição e análise do processo de teste que permitem avaliar a sua condução e questões de qualidade de software.

Embora o último nível seja o ideal, para atingi-lo é preciso melhorar cada fase do processo de teste de forma gradativa. Esta classificação (Figura 10) acaba servindo como orientação de que etapas priorizar na busca deste ideal.

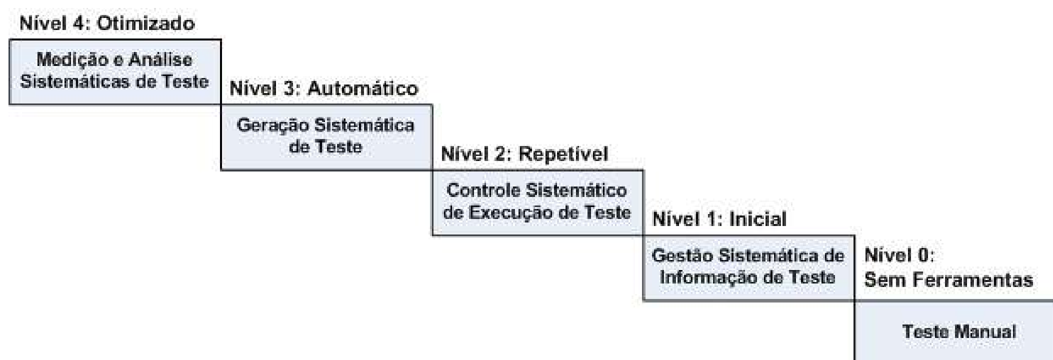

Figura 10 – Níveis de Automação de Teste (Tradução: Gao et al., 2003)

2.5.2. Geração Automática de Dados de Teste

Dentre os níveis de maturidade de automação, o que está relacionado diretamente à concepção dos testes de forma sistemática é o terceiro. Em meio às práticas apontadas, é de particular interesse de pesquisa avaliar um pouco mais detalhadamente a geração de dados de teste.

PUC-Rio - Certificação Digital Nº 0611931/CA

A construção manual de um conjunto de dados de teste representa uma grande parte do esforço de validação e verificação num projeto de software. Segundo levantamento feito por alguns estudos (Ince, 1987; Edvardsson, 1999), as técnicas utilizadas para automatizar esta construção são norteadas pela estrutura do código fonte da aplicação, pelas estruturas de dados existentes e pelas interfaces de software disponíveis.

Esses trabalhos relacionam alguns tipos de geradores: randômico, adaptativo, baseado em sintaxe, baseado em caminhos de execução, baseados em fluxo de dados, orientado por especificação etc. Destes geradores, o que é baseado em sintaxe será comentado a seguir.

2.5.3.
Gerador de Teste Baseado em Sintaxe

Um dado de teste pode ter sua estrutura expressa por meio de uma notação textual. Um exemplo de notação utilizada na descrição de dados de teste (Figura 11) é a BNF (*Backus-Naur Form*). Um gerador baseado em sintaxe utiliza a sintaxe desta notação como guia para a construção de casos de teste, gerando dados de teste que a satisfaçam. Portanto, não depende da estrutura do programa a ser testado e pode ser utilizado em testes funcionais de software.

PUC-Rio - Certificação Digital Nº 0611931/CA

```
<postal-address> ::= <name-part> <street-address> <zip-part>

     <name-part> ::= <personal-part> <last-name> <opt-jr-part> <EOL>
                   | <personal-part> <name-part>

 <personal-part> ::= <first-name> | <initial> "."

<street-address> ::= <opt-apt-num> <house-num> <street-name> <EOL>

      <zip-part> ::= <town-name> "," <state-code> <ZIP-code> <EOL>

   <opt-jr-part> ::= "Sr." | "Jr." | <roman-numeral> | ""

/* Fonte: U.S. Postal Address Description. */

/* Note que alguns termos não foram especificados como o formato de
'first-name', 'last-name', 'ZIP-code' entre outros.  Novas regras devem
ser adicionadas para contemplar estes casos. */

/* A regra que define 'name-part' é um exemplo de definição recursiva.
Embora pareça estranho, por esta regra uma pessoa pode ter vários
termos 'first-name', situação válida que retrata o primeiro nome
composto (ex. Ricardo Augusto). */
```

Figura 11 – Exemplo de Notação – BNF

Um dos problemas enfrentados por esta abordagem de geração de dados de teste são as dependências de contexto não retratadas por uma análise puramente sintática. Gerar dados corretos sintaticamente não garante uma semântica também correta na aplicação dos dados de teste.

Algumas soluções foram apontadas (Ince, 1987) para contornar essa situação. Uma delas é o uso de sintaxes dinâmicas. Neste caso, os dados são gerados inicialmente sem se preocupar com o contexto e depois são corrigidos levando em consideração as dependências de contexto. Outro procedimento é a adoção de gramáticas de atributos (Figura 12). As definições deste tipo de gramática são sobrecarregadas com atributos que representam informações contextuais associadas aos elementos da gramática. A partir deles é possível construir casos de teste completos com a descrição dos dados de teste e das saídas esperadas para um sistema em teste. Deste modo, gramáticas de atributos são usadas para expressar heurísticas de teste. A grande questão associada a esse tipo de gramática é se a complexidade da estrutura dos dados de teste não implicaria num crescimento muito rápido do tamanho da gramática.

```
number =  Nonfraction(bitstring)
          | Fraction(bitstring bitstring)
          [float value;  /* synthesized */ ];


bitstring = Oneb(bit)
            | Moreb(bitstring bit)
            [ float value;  /* synthesized */
              int length;   /* synthesized */
              int scale;    /* inherited */ ];

bit = One() | Zero()
      [ float value;  /* synthesized */
        int scale;    /* inherited */ ];

/* Fonte: D. Knuth, Semantics of Context Free Languages */
/* The abstract syntax tree of fractional binary numbers, attributed */
```

Figura 12 – Exemplo de Gramática de Atributos

As vantagens do uso deste tipo de gerador são forçar a documentação precisa da estrutura de dados de teste e produzir uma enorme quantidade de dados de teste. Ao assegurar que cada regra numa gramática seja exercitada e que uma distribuição estatística dos dados de teste seja obtida, este gerador permite que os testes aumentem de complexidade gradativamente.

PUC-Rio - Certificação Digital Nº 0611931/CA

A técnica adotada por geradores baseados em sintaxe apresenta, todavia, algumas limitações. Há classes de dados que não podem ser gerados a exemplo dos números primos. A escrita de regras sintáticas para conjuntos complexos de dados pode ser suficientemente entediante.

2.6. Teste Baseado em Modelo

O objetivo do teste de software é a detecção de falhas. As falhas equivalem às diferenças notadas entre os comportamentos da implementação e o que era esperado com base na especificação.

O teste baseado em modelo é uma variante de teste em que modelos explícitos são usados para capturar o comportamento de um sistema e de seu ambiente. Os pares de entrada e saída de um modelo de implementação são interpretados como casos de teste da implementação. A saída do modelo corresponde à saída esperada do sistema em teste.

Uma definição para o teste baseado em modelo (Utting et al., 2006) é o processo de derivação automática de casos de teste concretos a partir de modelos formais abstratos e execução dos casos de teste.

2.6.1. Características do Modelo de Teste

O modelo de teste precisa ser validado e os requisitos do sistema em teste nele figurados precisam ser avaliados quanto à consistência e à completeza. Isto implica o uso de um modelo mais simples que o sistema ou, pelo menos, de fácil verificação, modificação e manutenção. Do contrário, o esforço na sua elaboração não se justificaria.

O modelo deve levar em consideração os conceitos existentes. Quaisquer omissões no modelo corresponderão a partes não testadas com base no modelo proposto. Deve ser preciso o suficiente para permitir a geração de casos de teste sem ambigüidade e redundância. Isto torna útil a sua adoção e aumenta a confiabilidade dos testes.

PUC-Rio - Certificação Digital Nº 0611931/CA

2.6.2. Taxonomia de Teste Baseado em Modelo

Uma solução de teste, para ser compreendida e comparada com outras, deve adotar uma classificação segundo uma taxonomia de referência.

Uma taxonomia foi proposta (Utting et al., 2006) para o teste baseado em modelo. Ela identifica diferentes dimensões e suas possíveis instanciações, agrupadas pelos passos do metaprocesso de teste definido (seção 2.2).

Em relação à ***etapa de modelagem***, as dimensões são o ***escopo do modelo***, ***nível de redundância***, ***características do modelo*** e ***paradigma*** adotado. A primeira define se o escopo é o sistema em teste, o ambiente de teste ou os dois. A segunda reflete se o modelo é utilizado apenas para teste ou também para o desenvolvimento do sistema em teste. A terceira classifica o modelo segundo o grau de determinismo, questões de tempo e natureza de eventos (discreto; contínuo; híbrido). A última identifica que notação foi utilizada para modelar o comportamento do sistema a exemplo das notações baseadas em estado, transição de estados, histórico, funcionalidades, operações e fluxo de dados.

Quando se trata da ***etapa de geração de testes***, as dimensões aplicadas são o ***critério de seleção*** e a ***tecnologia de geração***. O critério de seleção pode envolver cobertura de estrutura do modelo, cobertura de dados, cobertura baseada em requisitos, especificação de caso de teste, cobertura baseada em defeito (ex. uso de mutantes). Em termos de tecnologia, são exemplos de instâncias: geração aleatória; algoritmos de busca em grafos; verificação de modelos; execução simbólica.

A última dimensão está associada ao ***tempo de execução dos testes***. Estes podem ser realizados de forma ***online***, requerendo que gerador e sistema de teste estejam simultaneamente em execução. Em alguns casos o próprio sistema pode apresentar uma estrutura de auto-teste. A outra possibilidade é a ***off-line***, quando os testes são gerados num momento e são executados posteriormente no sistema em teste sem a presença do gerador.

PUC-Rio - Certificação Digital Nº 0611931/CA

2.6.3. Processo de Teste Baseado em Modelo

Neste item, um processo genérico de teste baseado em modelo (Figura 13) é apresentado (Utting et al., 2006) de modo a ressaltar os pontos de modificação e extensão do processo de teste definido anteriormente, que pode ser considerado um metaprocesso de teste.

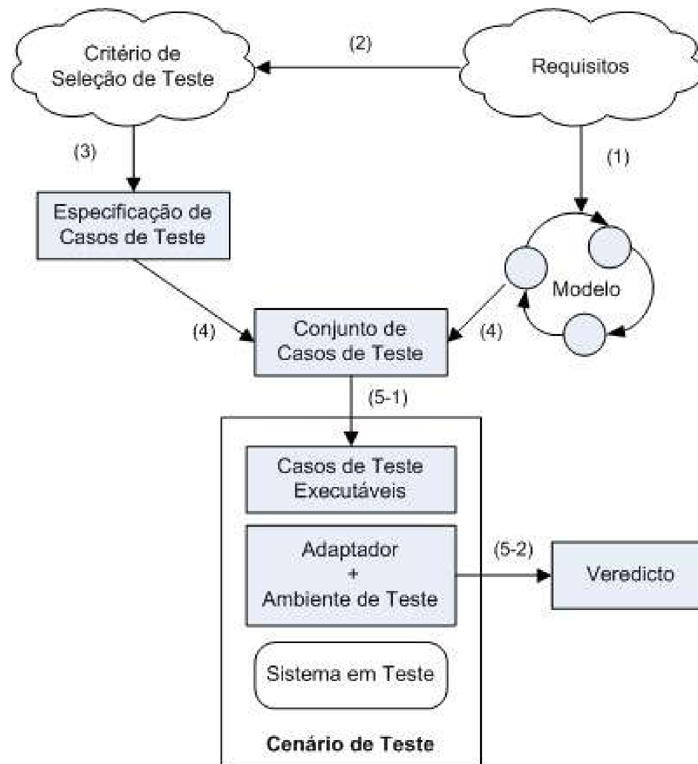

Figura 13 – Processo de Teste Baseado em Modelo (Adaptação: Utting et al., 2006)

O primeiro passo é a ***construção do modelo*** do sistema em teste com base na especificação de requisitos. Este modelo retrata o comportamento desejado para o sistema e poderá ser descrito em vários níveis de concretude. De acordo com o nível de concretude adotado, características funcionais e não funcionais serão avaliadas ou descartadas nos testes. Este passo equivale ao primeiro passo do metaprocesso de teste (Figura 14).

PUC-Rio - Certificação Digital Nº 0611931/CA

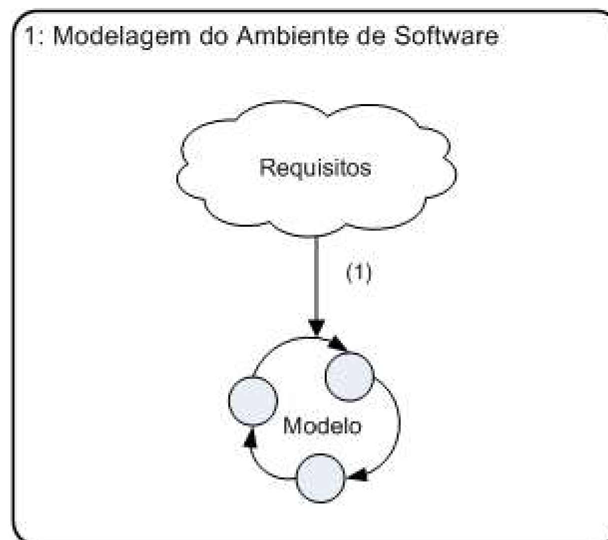

Figura 14 – Extensão do Primeiro Passo do Metaprocesso de Teste

O segundo passo é a ***definição do critério de seleção de teste***. Este critério define que comportamentos descritos pelo modelo serão avaliados. Descreve, por conseguinte, um conjunto de teste. O critério pode estar relacionado a funcionalidades do sistema, estruturas do modelo, formas de interação com o sistema, entre outros, assim como a um conjunto bem-definido de faltas.

O terceiro passo é a ***transformação do critério de seleção de teste em especificação de caso de teste***. Esta especificação formaliza o critério e o torna operacional. Dado um modelo e uma especificação de caso de teste, um gerador automático será capaz de elaborar um conjunto de teste. A especificação se diferencia do conjunto de teste pelo fato da primeira ser genérica e o segundo ser sua especialização. Neste passo, todos os testes são enumerados.

O quarto passo é a ***geração do conjunto de teste***. Nesta etapa, um grupo de casos de teste que satisfazem uma determinada especificação de caso de teste é gerado. Este grupo compõe o conjunto de teste.

O segundo, o terceiro e o quarto passos do processo de teste baseado em modelo correspondem à seleção dos cenários de teste do metaprocesso de teste (Figura 15).

PUC-Rio - Certificação Digital Nº 0611931/CA

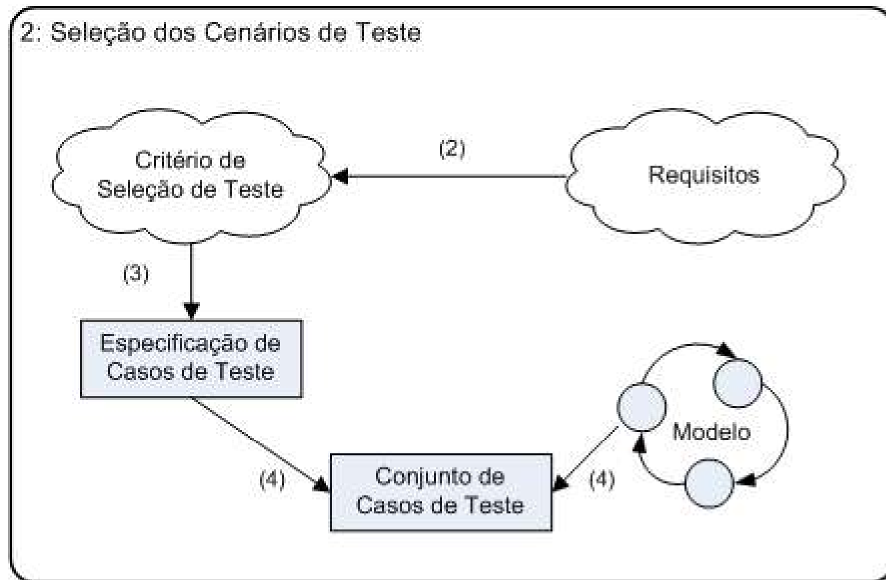

Figura 15 – Extensão do Segundo Passo do Metaprocesso de Teste

O quinto passo é a ***execução dos casos de teste***. Este passo é dividido em dois estágios. Corresponde à execução e avaliação dos cenários de teste no metaprocesso de teste (Figura 16).

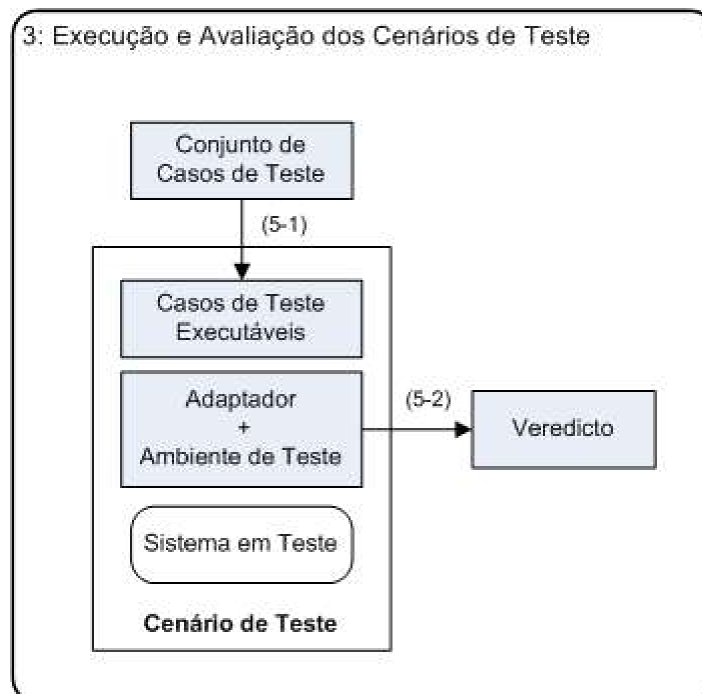

Figura 16 – Extensão do Terceiro Passo do Metaprocesso de Teste

PUC-Rio - Certificação Digital Nº 0611931/CA

No primeiro estágio, o objetivo é conciliar os diferentes níveis de abstração entre o modelo e o sistema em teste. Para isto, é preciso aplicar uma entrada de dados concreta ao sistema em teste e registrar a saída do mesmo. Um "adaptador" é usado para fazer esta ponte. Ele é um componente que concretiza a entrada de dados e o resultado esperado, compara resultados concretos e gera a análise.

No segundo estágio, um veredicto é gerado. Este é o resultado da comparação da saída do sistema com a saída definida pelo caso de teste. Um veredicto pode apresentar os seguintes valores: "passou", "falhou", "inconclusivo", "erro". Quando a saída esperada e a saída obtida estão em conformidade, o veredicto é "passou". Do contrário, é "falhou". Na situação em que não é possível avaliar a conformidade, então o resultado é "inconclusivo". Se o próprio dispositivo de teste apresentar erros (exceções) durante a sua execução, então o resultado é "erro".

PUC-Rio - Certificação Digital Nº 0611931/CA

2.6.4. Níveis de Concretude de Teste

Um ponto crítico para a geração e execução de casos de testes é compreender as diferenças de concretude entre a especificação dos casos de teste, o modelo de geração de casos de teste e a implementação do sistema em análise (Prenninger & Pretschner, 2005).

A especificação descreve os requisitos do sistema segundo as necessidades manifestadas por um ou mais interessados (*stakeholders*). Muitas vezes é feita de maneira informal e imprecisa, sendo assim mais abstrata. O modelo de teste descreve um conjunto de funcionalidades, propriedades e comportamentos deste sistema de maneira mais rigorosa e formal. O grau de detalhamento do modelo varia de acordo com os objetivos de teste, sendo alguns aspectos mais explorados que outros. A implementação é a realização (concretização) da especificação. Ela pode apresentar falhas decorrentes de interpretação equivocada de requisitos, requisitos incorretos ou de requisitos omitidos.

Uma especificação de teste deve ser detalhada o suficiente de maneira a obter resultados expressivos quando é comparada à implementação do sistema. Ela deve refletir adequadamente os requisitos do cliente. O processo responsável por garantir a conformidade entre artefatos produzidos e requisitos é a validação.

Os modelos de teste são elaborados com o objetivo de facilitar a validação. Quando um modelo de teste é tão específico como a implementação do sistema, não justifica o esforço de sua construção dado que a sua própria validação teria complexidade equivalente à da validação da implementação. Deste modo, os modelos devem ser resultados de uma simplificação e, logo, não capturam todos os atributos de sua representação original. Também são pragmáticos e por isso os modelos não estão relacionados de uma única maneira a sua representação. Portanto, normalmente são mais genéricos que a implementação do sistema.

Como conseqüência dos diferentes níveis de concretude, os casos de teste gerados não podem ser aplicados diretamente a uma implementação. As entradas aplicadas precisam ser descritas de forma concreta e compatível à implementação. As saídas do sistema, também concretas, são comparadas com o resultado esperado descrito pelos oráculos de teste dos casos de testes. Para que a comparação de resultados seja possível, os oráculos também precisam ser concretos. Esta verificação também deve levar em consideração o ambiente do sistema e sua influência na execução do mesmo.

PUC-Rio - Certificação Digital Nº 0611931/CA

2.7. Teste Dirigido por Modelo

Os diferentes níveis de concretude empregados na modelagem de teste e as variações de comportamento obtidas em virtude do ambiente do sistema em teste sugerem a adoção de modelos independentes de plataforma (*platform-independent models* – PIMs) e modelos específicos de plataformas (*platform-specific models* – PSMs). Desta forma, uma estratégia similar à de arquiteturas dirigidas por modelo (*model-driven architectures* – MDAs) seria utilizada no contexto de teste de software (Heckel & Lohmann, 2003; Dai 2004).

Seguindo uma estratégia deste tipo, o projeto de modelos de teste independentes de plataforma seria reutilizado e garantiria a interoperabilidade dos mesmos ao longo de várias plataformas. Caberia à parte do modelo específica de cada plataforma realizar a transformação necessária para execução correta dos testes no seu ambiente.

Segundo um levantamento feito (Heckel & Lohmann, 2003), muitas das abordagens de teste baseado em modelo não consideram a distinção entre modelos

independentes de plataforma e modelos específicos de plataforma. Estes modelos ou são projetados de acordo com uma plataforma específica ou são genéricos e representam apenas as informações independentes de plataforma.

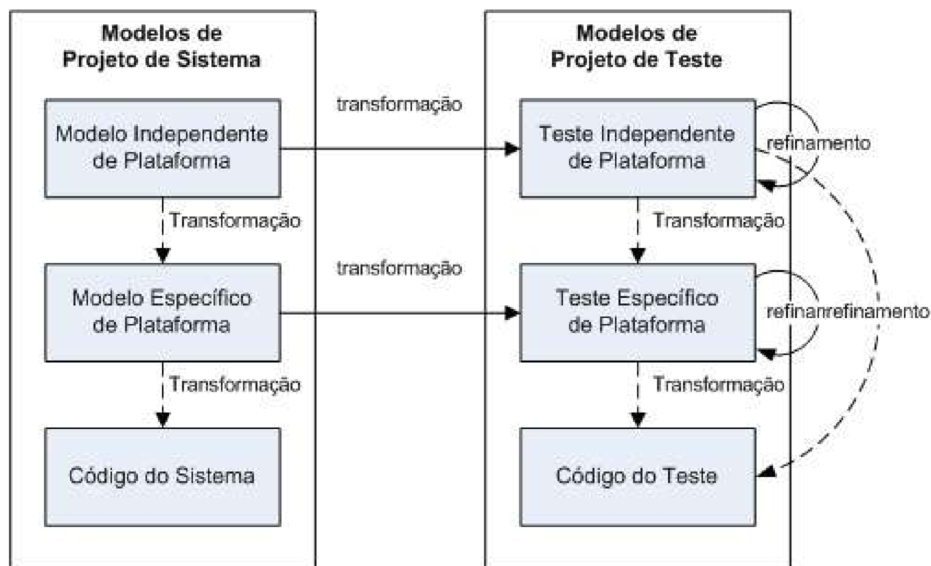

Figura 17 – Teste de Dirigido por Modelos (Adaptação: Dai, 2004)

Com o intuito de se beneficiar da separação de modelos na geração e execução de testes, a estratégia de ***teste dirigido por modelos*** (Figura 17) se propõe a ser o refinamento das principais atividades do teste baseado em modelo adotando essa estratégia estilo MDA.

As atividades diretamente afetadas por esse paradigma seriam a geração de casos de teste a partir de modelos de acordo com um dado critério de cobertura, a geração de oráculos de teste para determinar os resultados esperados de um teste e a execução de testes em ambientes de teste possivelmente gerados a partir de modelos (Heckel & Lohmann, 2003). As duas primeiras são independentes de plataforma, enquanto a última requer o uso de modelos específicos capazes de gerar os ambientes de teste e de mapear os casos de teste e oráculos genéricos na plataforma específica correspondente.

PUC-Rio - Certificação Digital Nº 0611931/CA

2.8. Desafios

Os processos, técnicas, ambientes e ferramentas de teste sofrem adaptações às mudanças de contextos e tecnologias existentes e evoluem de acordo com a consolidação dos conhecimentos adquiridos na área de teste de software. Nesta linha de evolução, novos e recorrentes problemas e barreiras precisam ser transpostos para que objetivos maiores sejam alcançados. Um estudo recente destaca alguns dos desafios a serem superados pelas linhas de pesquisa da área (Bertolino, 2007). Estes desafios podem ser divididos segundo os aspectos de modelagem, automação e eficiência. No texto a seguir, os desafios enumerados estão diretamente relacionados a esta pesquisa.

2.8.1. Desafios à Modelagem

PUC-Rio - Certificação Digital Nº 0611931/CA

Uma questão sempre recorrente na área de testes é como combinar diferentes estilos de modelagem tais como os baseados em assertivas, em contratos, em transições de estado ou em gramáticas. Como cada modelo é uma interpretação diferente da realidade observada, é provável que o uso articulado de modelos diferentes seja uma abordagem mais completa, embora mais complexa.

Outro ponto é a necessidade de integrar a prática de teste baseado em modelo aos processos de software atuais, minimizando os impactos nas atividades de desenvolvimento. O desafio é tornar os modelos de teste tão genéricos quanto possível, preservando a sua capacidade de gerar casos de teste executáveis e mantendo a rastreabilidade de requisitos aos testes durante o ciclo de vida do sistema.

Algumas dificuldades estão relacionadas aos oráculos de teste. É preciso conciliar estados e comportamentos especificados de forma abstrata com estados e comportamentos observados de forma concreta. Existe ainda uma decisão a ser tomada quanto ao balanceamento entre a precisão e o custo dos modelos. Modelos muito detalhados facilitam a geração dos testes, mas são de difícil elaboração e validação. A expressividade e a eficiência dos modelos adotados também devem ser medidas e contrastadas de acordo com ambiente de aplicação. A falta de ortogonalidade entre a seleção de casos de teste e a definição dos

oráculos de teste é outro fator a ser analisado. Atualmente, ambos são derivados de um mesmo modelo.

2.8.2. Desafios à Automação

A automação de testes derivados de modelo ainda não é uma solução madura e carece de um maior número de estudos de caso para ter seus benefícios constatados (Bertolino, 2007). Algumas extensões deste paradigma de testes ainda precisam ser postas à prova como o caso de testes baseados em modelos de domínios específicos. Estes seriam capazes de restringir ainda mais o escopo de testes. Estudos precisam ser feitos para averiguar se a especialização dos mecanismos de testes seria benéfica à automação.

O campo das idéias é fértil e podem ser tão avançadas quanto a sua imaginação permitir. No que tange à automação, testes *online* poderiam ser concebidos de forma a verificar a corretude de um sistema durante sua própria execução, portanto, uma análise dinâmica.

2.8.3. Desafios à Eficiência

Muito se questiona sobre o comportamento do teste baseado em modelo quando se tem um ganho de escala. O quanto ele será capaz de atender a demanda sem aumentar a complexidade de modelagem e sem perder a eficiência.

Dada a hipótese de que este tipo de teste atenda as necessidades de validação e verificação de sistemas, como estimar o custo para sua adoção passa a ser um critério de adequação à realidade do ambiente de desenvolvimento.

O controle de evolução dos testes tem impacto direto sobre essa questão de custos. Adotando técnicas de regressão é possível reaproveitar casos de testes previamente executados e diminuir os custos da produção de novos. Contudo, ainda é necessário elaborar ferramentas de manutenção dos testes mais eficientes. O uso de padrões de teste também seria uma forma de reuso de conhecimento aplicado a testes que poderia aumentar a eficiência e reduzir custos.

PUC-Rio - Certificação Digital Nº 0611931/CA