MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INVESTIGAÇÃO SOBRE MÉTODOS DE GERAÇÃO DE SEQUÊNCIAS DE TESTE EM GRAFOS BALANCEADOS

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA
(PIBIC/CNPq/INPE)

Matheus Monteiro Mariano (FATEC, Bolsista PIBIC/CNPq)
E-mail: matheus.mariano2@fatec.sp.gov.br

Nandamudi Lankalapalli Vijaykumar (LAC/CTE/INPE, Orientador)
E-mail: vijay.nl@inpe.br

COLABORADORES

Érica Ferreira Souza (UTFPR)
André Takeshi Endo (UTFPR)

Julho de 2016

1. INTRODUÇÃO

Testes de Software têm se tornado relevantes por garantir a qualidade no sistema desenvolvido. Os testes podem ser classificados como de Testes Caixa Preta, por dependerem de um modelo do software a ser desenvolvido, ou Testes Caixa Branca, por se basearem no código. O trabalho descrito neste relatório final de Iniciação Científica explora Testes Caixa Preta focando na melhoria de desempenho do critério *Switch Cover*.

1.1 MOTIVAÇÃO

A construção de um software é, sem dúvida, um grande desafio. Erros em sistemas considerados críticos e complexos podem afetar a produtividade da empresa, bem como a qualidade final do produto gerado (Delamaro et al., 2007). Problemas acontecem por causa da grande quantidade de código, processos, métodos e rotinas necessárias para satisfazer as necessidades de resolução de um problema em um determinado software. Portanto, o ideal é realizar testes nestes sistemas para tentar encontrar o maior número de problemas possíveis, e então aplicar uma solução.

Porém, é inviável descobrir um conjunto de testes que seja ideal para verificar todos os caminhos do software, pois, geralmente, dependendo da natureza do teste, a sequência gerada pode ser potencialmente infinita, o que excede a capacidade de teste (Black, 2008) ou, no mínimo muito grande, tornando-se impraticável (Rapps et al, 1985). Dessa forma, o ideal é escolher casos de teste que diminuam estes valores em poucos conjuntos (Delamaro et al, 2007). Apesar da geração de testes trazer um benefício e segurança ao sistema, ela também estabelece um padrão de garantia da qualidade do sistema (Endo, 2010), sendo importante quando aplicada desde o início do desenvolvimento, pois isso evita que na implementação se propague erros ao programa, tornando assim mais fácil a sua remoção (Souza, 2010).

Contudo, a geração de casos de testes ainda é um processo manual que possui pouco reuso (Apfelbaum, 1997), pois um dos grandes problemas em criar casos de teste é descobrir uma estratégia para encontrar casos de testes que sejam válidos e confiáveis. Com isso, o uso da modelagem torna-se muito econômico, pois captura a essência do conhecimento do sistema e, ao usar esse conhecimento de forma adequada, faz a criação de casos de teste prosperar,

podendo o time de desenvolvimento captar essa informação e analisar a estrutura do sistema de forma eficaz. Esta abordagem se tornou conhecida como Teste Baseado em Modelo (TBM).

Nesta abordagem, TBM, o comportamento do software deverá ser representado. Devido à semelhança com os grafos, as Máquinas de Estados Finitos (MEF) (Lee & Yannakakis, 1996) se tornaram populares onde há um conjunto de estados e arcos de transição entre estes estados. Os arcos de transição possuem rótulos que são conhecidos como eventos ou entradas. A mudança de um estado para outro ocorre quando este evento ou entrada é estimulada. Por isso a MEF é conhecida em representar sistemas reativos, onde estes reagem ou respondem a estímulos. A resposta é a mudança de um estado para outro. Mas ainda há uma pergunta: como são gerados casos de testes, uma vez que o comportamento do software esteja captado ou representado por uma MEF. Como uma MEF se assemelha com um grafo, a partir da teoria de grafos, há algoritmos que percorrem a MEF gerando as seguintes informações: estado atual, evento ou entrada, e próximo estado. Se organizar estas informações partindo de um estado qualquer e retornar a este mesmo estado, então, tem-se uma sequência ou caso de teste.

Quando o software é representado por uma MEF, há diversos métodos ou critérios de teste que geram as sequências: Transition Tour, UIO (Unique Input/Output Sequence), DS (Distinguishing Sequence) (Sidhu, 1989), Switch Cover (Pimont & Rault, 1976). Um destes critérios, o *Switch Cover*, também conhecido como um teste de “todas as combinações” por definir que todos os pares adjacentes sejam passados pelo menos uma vez, será o foco deste relatório e o utilizado para este trabalho.

1.2 PROBLEMA

O critério *Switch Cover* parte de uma MEF, a qual é convertida para um grafo dual. Este grafo dual tem que ser balanceado a partir do qual é realizado um ciclo Euleriano para gerar as sequências ou casos de testes. Embora o critério *Switch Cover* seja antigo e possua diversas versões implementadas (Arantes et. al, 2014; Martins et. al, 1999; Pimont et al, 1976), tais implementações apresentam alguns problemas observados ao longo dos anos. Alguns exemplos podem ser citados como algumas execuções do *Switch Cover* não conseguirem tratar adequadamente MEFs complexas; inconsistências entre o conjunto de

casos de teste gerados e a MEF, uma vez que os casos de teste não correspondem fielmente ao modelo da Máquina de Estado Finito; ou até o grande esforço computacional para lidar com as duas principais etapas do critério *Switch Cover*, que são: o balanceamento do grafo dual, que é a transformação das arestas da MEF em nós, para realizar o ciclo Euleriano; e a geração dos casos de teste a partir deste novo grafo balanceado, que torna o processo extensivamente maior pela quantidade de caminhos extras criados. Dessa forma, o objetivo do trabalho desta Iniciação Científica é melhorar o desempenho do método Switch Cover explorando mecanismos de geração de buscas em grafos e, por tanto, gerar sequências de teste sem a necessidade do balanceamento do grafo dual.

1.3 PROPOSTA DE SOLUÇÃO

Este trabalho tem como objetivo investigar o critério *Switch Cover* utilizando o métods de **Busca em Largura** no grafo dual antes do processo de balanceamento, visando melhorar a eficiência e esforço computacional do critério. Com a MEF já criada em memória, tendo o seu grafo dual gerado a partir da MEF original, realiza-se esta busca para percorrê-lo e gerar sequências de testes. Para validar esta abordagem, os resultados desta serão comparados por métodos tradicionais que geram sequências ou casos de teste por meio do *Switch Cover*, como o processo de balanceamento e a geração do ciclo Euleriano. No final, como resultado, serão geradas duas sequências distintas, o que irá permitir realizar comparações como tempo de criação, quantidade de eventos de cada teste, e quantidade de sequências geradas, o que possibilitará determinar se a criação das sequências de testes antes da etapa do balanceamento é melhor ou não.

Este relatório está organizado da seguinte forma: O Capítulo 2 trata de um resumo da fundamentação teórica para contextualizar o trabalho desenvolvido nesta Iniciação Científica. A nova abordagem será descrita no Capítulo 3, seguido no Capítulo 4 dos resultados preliminares. O Capítulo 5 finalmente conclui o trabalho além de mostrar como o trabalho poderá ser continuado.

2. FUNDAMENTAÇÃO TEÓRICA

O objetivo deste capítulo é apresentar conceitos breves sobre teste de software e Testes Baseado em Modelo. Ele está dividido nas seguintes seções: 2.1 apresenta o conteúdo geral sobre Teste de Software, abrangendo Teste de Caixa Branca e Preta; enquanto que a seção 2.2 exibe conteúdos de Teste Baseado em Modelo, dividindo em seções por Máquinas de Estado Finito (MEF), critérios do Teste e, no capítulo 2.4 mais detalhadamente o critério *Switch Cover*.

2.1 TESTE DE SOFTWARE

Teste de Software é um processo de execução com a intenção de encontrar erros (Myers et al, 2011). Estes erros com o sistema podem aparecer de diversas formas, sendo a principal causa o erro humano. Apesar de softwares com o propósito de vasculhar erros não garantirem 100% de certeza da falta destes, há mecanismos que podem auxiliar o programador a criar um sistema menos suscetível a falhas (Delamaro et al, 2007).

Dentre as técnicas, existem diversos modelos de teste que tentam dar ao desenvolvedor meios para garantir a qualidade do seu software. O Teste Caixa Branca, também conhecido como Teste Estrutural, tem como princípio olhar o comportamento do sistema, baseado no código implementado. Já o Teste Caixa Preta, que é o foco neste trabalho, também conhecido como Teste de Entrada e Saída, é uma técnica que imagina o software como uma “caixa preta”. Os testes são inseridos sem saber o que há por dentro do algoritmo, pois são verificados apenas suas saídas e comparadas com um “resultado esperado”, que foi previamente especificado.

2.2 TESTE BASEADO EM MODELO

O Teste Baseado em Modelo (TBM) é a descrição de um modelo de como funciona ou se comporta um software, modelo este que é desenvolvido a partir da documentação e especificação do sistema (Dalal, S.R., et al; 1999). Este modelo é então usado para alguns métodos de geração dos casos de testes, como ilustrado na Figura 2.1. Dessa forma, o código

se torna dispensável na criação dos casos de teste, pois é a partir da documentação que será criado um modelo. Então, mesmo que o algoritmo esteja errado, o teste deve supor que o modelo está correto.

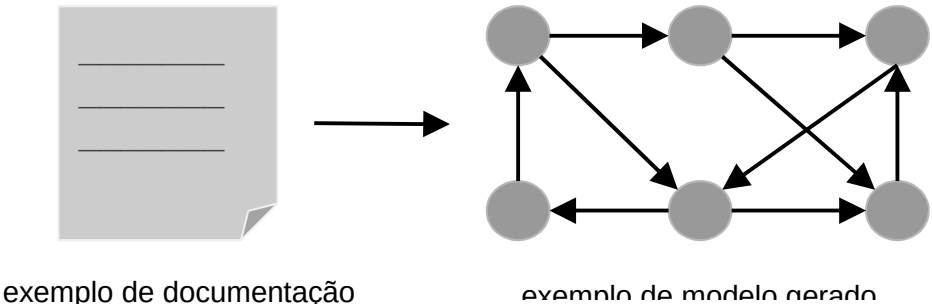

Figura 2.1 Teste Baseado em Modelo a partir da Documentação

Como a abordagem TBM depende de um modelo, as Máquinas de Estados Finitos (MEF) (Lee & Yannakakis, 1996) se tornam candidatas para representar o comportamento de um software. Uma descrição resumida de uma MEF será apresentada na Seção 2.3.

2.3 MÁQUINA DE ESTADO FINITO

As Máquinas de Estado Finito (MEF) (Lee; Yannakakis, 1996) são um conjunto de estados e arcos de transição entre estes estados. Os arcos de transição possuem rótulos que são conhecidos como eventos ou entradas. A mudança de um estado para um outro ocorre quando este evento ou entrada é estimulada. Para este trabalho, o modelo de Diagrama de Transição de Estados é o que será utilizado para visualizar uma MEF. Os círculos são os estados e as linhas entre eles as transições, e as setas a indicação do destino destas transições de um estado para o outro.

Figura 2.2 Representação de um siste de semáforo por meio de uma MEF

Um dos exemplos mais clássicos de uma MEF é a de um sistema de semáforo. Neste exemplo mostrado na Figura 2.2, foram especificados 4 estados: o início do processo; a troca

de cores, que irá disparar a mudança entre vermelho, amarelo e verde; o espera, que irá realizar um processo de espera no sistema; e o fim, quando é desligado. Esses processos possuem 5 transições, onde inicialmente o semáforo inicia o processo, realiza a troca de estado três vezes, com pausas neste meio tempo retornando ao mudança de estado, e depois indo ao início novamente para reiniciar o processo. Isso será realizado em *looping* até que por algum motivo o sistema precise ser desligado, e quando isto ocorrer irá para o estado fim. A MEF, como tal, especifica apenas o caminho; como será realizado e a quantidade de vezes a ser passado entre as transições é a partir da interpretação do sistema.

Apesar das Máquinas de Estado Finito trazerem uma informação completa ao testador, onde todas as possibilidades estão descritas através de seu diagrama, é difícil poder coletar esta informação sem se basear em algo que investigue os estados e transições. Então, mesmo que tenha em mãos o sistema, é inviável utilizar todas as entradas possíveis de uma máquina. Com isso, MEFs possuem diversos critérios que foram investigados para auxiliar o testador a explorar a MEF, e é possível encontrar na literatura diversos trabalhos que demonstram critérios para percorrer uma MEF. Um exemplo são critérios DS e UIO que estão implementados na ferramenta *WEB-PerformCharts* (Alessandro et al, 2014). No subcapítulo 2.5 será detalhado o critério *Switch Cover*, que é o utilizado por este trabalho (Sidhu, D.P. & Leung, T., 1989).

2.4 CRITÉRIO SWITCH COVER

O critério *Switch Cover*, também chamado de “todas as combinações” (Pimont & Rault, 1976), é um dos mais complexos na literatura, pois especifica que todos os pares de transições adjacentes devem ser executadas pelo menos uma vez. Para implementá-lo, o primeiro passo é transformar todas as transições de uma MEF em vértices de um grafo, pois assim torna o algoritmo menos complexo de analisar e tem uma maior amplitude em todos os pares de transições (Pimont & Rault; 1976), criando assim o grafo dual. Para exemplificar, a Figura 2.3 representa uma MEF simples, e a Figura 2.4, representa os vértices criados a partir das transições deste primeiro grafo.

Figura 2.3 Representação de uma MEF em um modelo
Fonte: Amaral (2005)

Figura 2.4 Grafo dual a partir da MEF original

Depois é necessário criar as arestas neste grafo dual. Para isto, é analisado o comportamento do grafo original, e verifica-se o caminho percorrido entre estes estados. Na Figura 2.5 mostra o grafo original onde destaca um caminho a partir do conjunto de um par de transição, enquanto que a Figura 2.6 exibe o grafo dual já preenchida de arestas mas, em destaque, a aresta que foi resultado da transição destacada do grafo original.

Figura 2.5 Nova transição no Grafo dual pela MEF original

Figura 2.6 Criando a nova transição

Agora deve-se balanceá-lo, analisando as polaridades dos vértices. Balanceamento consiste em duplicar as arestas de tal forma que o número de arestas que chegam deve ser igual ao número de arestas que deixam o vértice. Na prática, é fazer com que cada vértice possua o mesmo número de entradas e saídas, o que no final permite ter um caminho euleriano, onde cada aresta no grafo é visitada apenas uma vez (Lipschutz & Lipson, 1997). Um exemplo desta mudança pode ser vista na Figura 2.7 no vértice r, onde possui duas arestas de entrada, e uma de saída.

Figura 2.7 Grafo dual balanceado

Por fim, basta apenas gerar as sequências de teste percorrendo o grafo dual, iniciando de algum vértice inicial e voltando para ele próprio. Como o estado W na MEF original é o estado inicial, então tanto o vértice *a* quanto *c* do grafo dual serão iniciais. As sequências geradas para este grafo são: abf, abrbf, cf & crbf.

2.5 CONSIDERAÇÕES FINAIS

Houve diversas tentativas para aprimorar o critério de teste *Switch Cover* devida a sua complexidade. É visto que o balanceamento requer muita carga da máquina para MEFs complexas devido a criação de novas arestas para tornar os caminhos balanceados, desta

forma foi percebido que, ao tratar as MEFs como grafos, pode-se aplicar conceitos de buscas em grafos (como **Busca em Largura**) e realizar sequências de teste. Então neste trabalho, propomos mais uma abordagem para este critério, visando gerar as sequências de teste no processo sem a necessidade de balanceamento do grafo e nem gerar ciclo Euleriano.

3. MECANISMO ALTERNATIVO PARA MELHORIA DE DESEMPENHO DO CRITÉRIO *SWITCH COVER*

Como já foi mencionado, o critério *Switch Cover* é um dos mais antigos na literatura, sendo considerado como um dos mais difíceis por realizar uma análise que percorre todos os caminhos possíveis dentro de uma MEF. Sua principal etapa está na transformação da MEF em um grafo dual, onde suas transições se tornam vértices. Desta forma, com sua conversão em grafo, ela pode ser analisada por diversos algoritmos de buscas que percorrem um grafo para gerar sequências de teste. O Switch Cover utiliza o ciclo Euleriano, onde cada vértice do grafo possui um grau par, garantindo que sempre que entrar em um vértice através de uma aresta de entrada, ele possa sair por uma de saída. Porém, para isto, é necessário balancear o grafo dual. No entanto, este é um processo custoso, tanto o balanceamento quanto o processo de eulerisar o grafo pois acaba criando novos caminhos que não existem no grafo original, onde para cada nova aresta feita deverá verificar se todos os outros estão balanceados. Além disto, as novas arestas podem se tornar extensivas, principalmente para MEFs complexas, o que pode causar um esforço computacional significativo.

Figura 3.1 Visão geral do projeto

Desta forma, a nova proposta é criar sequências de teste com o grafo dual desbalanceado através de outros algoritmos de busca, como **Busca em Largura**. Esta nova

abordagem será comparada com o método tradicional que utiliza grafo dual balanceado e o ciclo Euleriano. A comparação será baseada na quantidade de sequências de teste geradas, quantidade de eventos, e o tempo gasto. A Figura 3.1 representa de forma simplificada a visão geral do projeto, enfatizando em passos as áreas mais relevantes.

De forma geral, na Figura 3.1, primeiramente o sistema identifica uma MEF em um diretório especificado. A MEF pode estar representada tanto em formato XML quanto em um formato em texto. Com a MEF inserida em memória (*step 1*), será aplicado o critério *Switch Cover* e criará o grafo dual (*step 2*). Com isso, serão geradas sequências de caso de teste a partir da **Busca em Largura** deste grafo dual (*step 3*) e adicioná-los em um arquivo cada um. Depois, em outra rodada, o processo de criação da **Busca em Largura** será trocada pelo processo de balanceamento neste grafo dual (*step 4*) e, então, uma nova sequência de teste através do ciclo euleriano (*step 5*), que também é salva em um arquivo. Por fim, através dos resultados salvos em dois arquivos separados, um programa de análise irá gerar os resultados deste análise.

Para este trabalho, foi implementado o algoritmo de **Busca em Largura** para os grafos não balanceados. A **Busca em Largura** é implementada verificando os vértices filhos a partir de um vértice pai, e visita-os caso não tenha sido visitado. Este algoritmo foi implementado da esquerda para à direita, cobrindo as arestas. Desta forma, considerando a MEF da Figura 3.2 e transformando-o em grafo dual, é gerado o grafo da Figura 3.3. Pode-se verificar que, ao explorá-lo, ela irá se comportar como o grafo da Figura 3.4.

Figura 3.2 Uma simples MEF

Figura 3.3 Grafo dual gerado pela MEF

Figura 3.4 Árvore da Busca em Largura gerado a partir do grafo dual

Para encontrar as sequências de teste, percorre-se a árvore gerada pela **Busca em Largura** a partir do nó inicial até um nó ser folha, salvando a sequência percorrida como uma sequência de teste. Desta forma, verificando a árvore da Figura 3.4, as sequências de teste geradas são: #1-abfa, #2-abfcf, #3-abfcr, #4-abrb.

Para provar a validade deste algoritmo, o Capítulo 4 mostra uma comparação das sequências de teste gerada pela MEF entre a **Busca em Largura** e o ciclo euleriano com o processo de balanceamento, expondo os resultados em termo de eficiências de tempo e quantidade de casos de teste.

4. RESULTADOS PRELIMINARES

Para esse estudo foram consideradas 2000 MEFs geradas de forma aleatória. Foram geradas MEFs com diferentes configurações e quantidades de estados, variando de 4 a 20 estados, onde se manteve fixa a quantidade de 4 transições para cada estado. Nota-se que foram desconsideradas MEFs que não sejam fortemente conectadas e todas as análises partem de um único estado inicial. Para o padrão dos gráficos, a linha azul representa a **Busca em Largura**, enquanto que a vermelha o critério *Switch Cover*.

Para as MEFs reais, foi utilizado neste trabalho o modelo do software do projeto *Software of the Payload Data Handling Computer* (SWPDC). O SWPDC é um software que gerencia a aquisição de dados científicos, de diagnóstico, a partir de câmeras de raio X (*Event Pre-Processor* (EPP) H1 e EPP H2) e um Sistema de Computação denominado Central Electronics Unit (CEU). O software SWPDC é composto de 20 (vinte) modelos para teste, ou seja, foram utilizados 20 (vinte) cenários representados por MEFs para a geração dos casos de teste (Santiago et al., 2010). Para fins de ilustração, apenas o cenário 03 será apresentado na Figura 4.1.

Figura 4.1. Modelo Statecharts do cenário 03 do software SWPDC.

Os resultados das MEFs aleatórias em termos de número de casos de teste são demonstrados na Figura 4.2. Nota-se que a **Busca em Largura** cobre uma larga escala em comparação ao Euleriano. Isto era esperado porque o critério *Switch Cover* parte de um estado inicial e deve retornar ao mesmo. Isto não quer dizer, porém, que o Switch Cover é melhor. A **Busca em Largura** gera um maior número de sequências de casos de teste porque o algoritmo cria uma árvore de expansão do grafo partindo de um raiz nó, até um nó folha - ou seja, não possuir nenhum filho. Este caminho do nó inicial até uma folha se torna uma sequência de teste, e é por este motivo que a **Busca em Largura** irá gerar mais sequências de teste. Enquanto isso, o *Switch Cover* gera uma única sequência com um tamanho grande que cobre apenas um grafo balanceado, que gerou novas arestas que não existiam no grafo original - apesar de estas arestas existiram no grafo, o que as tornam duplicadas.

Figura 4.2 Gráfico mostrando a relação dos dados em termo de número de casos de teste

Outro critério de análise usado foi em relação ao número entradas e eventos de teste. O resultado pode ser visto na Figura 4.3. Se analisarmos o número de entradas, o *Switch Cover* leva uma pequena vantagem, tendo um maior número de eventos comparado à **busca em largura**.

Figura 4.3 Gráfico mostrando os dados em termo de eventos

Em relação ao tempo, visualizável na Figura 4.4, claramente a **Busca em Largura** consegue uma larga vantagem em comparação ao *Switch Cover*, sendo completado em bem menos minutos. O problema do critério *Switch Cover* se deve ao balanceamento, pois o processo de verificação se o grafo está balanceado precisa ser feito a cada nova aresta inserida. Além disto, os casos de teste acabam se baseando em um grafo extenso, que possui arestas à mais, o que torna todo o processo mais longo. Note que os dois algoritmos foram rodados 100 vezes para cada MEF e tirado uma média de tempo. O tempo foi representado em milissegundos.

Tabela 4.4 Gráfico mostrando a relação de tempo em milissegundos

As análises, em relação às MEFs reais do SWPDC, podem ser vistas a seguir. Na Figura 4.5, o resultado do maior número de casos de teste se manteve o mesmo da MEF aleatória. Porém, em relação ao número de eventos, a **Busca em Largura** conseguiu possuir uma vantagem em relação ao *Switch Cover*, com quase metade do valor de diferença. Já no tempo, ambos geraram o mesmo resultado, porém nota-se que que o critério *Switch Cover* foi um pouco mais lento do que a busca em largura desta vez, em vez da grande diferença como vista nas MEFs aleatórias.

Figura 4.5 Gráfico comparando a Busca em Largura com Switch Cover pela MEF

5. CONCLUSÕES E TRABALHOS FUTUROS

Como mencionado nos capítulos anteriores, o *Switch Cover* é um critério muito empregado para gerar sequências de teste. No entanto, devido ao balanceamento do grafo e aplicação do ciclo Euleriano, dependendo da complexidade da MEF, este processo se torna lento e ineficiente. Neste sentido, como uma primeira tentativa, foi proposto o algoritmo de **busca em largura** visualizando a MEF como se fosse uma árvore. Os resultados se mostraram satisfeitos e em alguns parâmetros se torna um pouco mais eficiente do que o *Switch Cover*. Isso gera uma motivação de investigar técnicas alternativas para contornar os problemas de esforço computacional do método *Switch Cover*.

Como trabalhos futuros, pretende-se realizar pesquisas com outros algoritmos de busca em árvore, como busca em profundidade e a sua implementação e consequente validação. Com isto, pretende-se conduzir uma análise comparativa entre os resultados deste trabalho com as das geradas pela busca em profundidade. Também pretende-se testar outras MEFs conhecidas para validar tanto o código utilizado neste trabalho, quanto o que será projetado para outras buscas. Além disto, espera-se incorporar, após os resultados da conclusão deste trabalho, os critérios novos gerados na ferramenta *WEB-Performed Charts*.

6. REFERÊNCIAS

Amaral, A. S. M. S. Geração de casos de testes para sistemas especificados em Statecharts. 162 p. Dissertação (Mestrado) — Instituto Nacional de Pesquisas Espaciais, (INPE-14215-TDI/1116). Dissertação (Mestrado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2005. Disponível em: <http://urlib.net/rep/sid.inpe.br/MTC-m13@80/2006/02.14.19.24?languagebutton=pt-BR>.

Apfelbaum, L., & Doyle, J. Model Based Testing. Software Quality Week Conference, 1997.

Arantes, A.O.; Santiago Júnior, V.A.; Vijaykumar, N.L.; Souza, E.F. Tool Support for Generating Model-Based Test Cases via Web. International Journal of Web Engineering and Technology. 9(1), pp. 62-96. 2014. ISSN:1476-1289.

Black, R. Advanced Software Test Design Techniques State Diagrams, State Tables, and Switch Coverage. RBCS: Rex Black Consulting Services, Advanced Software Testing: Volume 1, 2008.

Chow, T.S. Testing software design modeled by nite-state machines. IEEE Transactions on Software Engineering, SE-4(3), 1978.

Dalal, S.R., et al. "Model-based testing in practice." Proceedings of the 21st international conference on Software engineering. ACM, 1999.

Delamaro, E.; Maldonado, J. C.; Jino, M. Introdução ao Teste de Software. Rio de Janeiro: Ed. Elsevier 2007.

El-Far, K.I., & Whittaker A.J. Model-based Softwares Testing. Florida Institute of Technology, Encyclopedia on Software Engineering, 2001.

Endo T.A. Teste baseado em máquinas de estados finitos para aplicações orientadas a serviço. São Paulo: Serviço de Pós-Graduação do ICMC-USP, 2010.

Lee, D, & Yannakakis, M. Principles and methods of testing finite state machines: a survey. Proceedings of the IEEE 84(8):1090–1123, (1996).

Marucci, R.A., et al. OORTs/ProDeS: Definição de Técnicas de Leitura para um Processo de Software Orientado a Objetos. Simpósio Brasileiro de Qualidade de Software, v. 1, 2002.

Myers, G.J.; Sandler, C., and Badgett, T. The art of software testing. John Wiley & Sons, 2011.

Pimont, S., & RAULT, C. J. A Software Raliability Assessment Based on a Structural and Bheaviiral Analysis of Programs. THOMSON - CSF, 1976.

Rapps, S., & Weyuker, E.J. (1985). Selecting software test data using data flow information. IEEE transactions on software engineering, (4), 367-375.

Sidhu, D. P., & Leung, T. (1989). Formal Methods for Protocol Testing: A Detailed Study. Transactions on Software Engineering. IEEE Computer Society. Dept. of Comput. Sci., Maryland Univ., Baltimore, MD. 13(4), 413-426.

Santiago, V.A.; Cristini, A.M.; Vijaykumar, N.L. Model-based test case generation using Statecharts and Z: A comparison and a combined approach. São José dos Campos, 2010. 72 p. (INPE-16677-RPQ/850). Disponível em: <http://urlib.net/sid.inpe.br/mtc-m19@80/2010/02.26.14.05>.

Souza, F.E. Geração de Casos de Teste para Sistemas da Área Espacial usando Critérios de Teste para Máquinas de Estados Finitos. INPE-16682-TDI/1627, 2010.