Critérios para geração de casos de teste baseados em modelo descritos em PcML

Beatriz Aruk da Silva[1], **Lucas Tosetto Teixeira**[1] **e Gian Ricardo Berkenbrock**[1]

[1]Universidade Federal de Santa Catarina (UFSC)
Centro Tecnológico de Joinville
Caixa Postal 89.219-600 – Joinville – SC – Brazil

beatrizsaruk@gmail.com, tosetto.lucas@gmail.com, gian.rb@ufsc.br

Abstract. *One of the main steps during software development is testing and verifying the implemented functionalities. Testing software can be a task that needs a lot of effort; even so, it does not guarantee an error will be detected. The automation of test generation comes up as an approach to support software verification and validation. Nowadays, there are different models for generating test cases, based on finite state machines (FSM), which provide support during the software test. Each model can travel the FSM differently, providing different test cases. This study aims to compare three different test criteria, Switch Cover, Unique Input/Output (UIO), and Distinguishing Sequence (DS), for four FSMs and evaluate the test cases for each one, analyzing their cost.*

Resumo. *Uma das principais etapas durante o desenvolvimento de um software é o teste e verificação das funcionalidades implementadas. Testar um software pode ser uma tarefa que demande grande esforço e mesmo assim não garante que um erro seja detectado. A automatização da geração de testes surge como uma abordagem que auxilia a verificação e validação de softwares. Atualmente existem diferentes modelos de critérios para geração de casos de testes, com base em máquinas de estados finitos (MEF), que dão suporte durante a etapa de testes de software. Cada modelo pode percorrer a máquina de estados finitos de uma maneira, entregando diferentes casos de testes. Este estudo pretende comparar três diferentes critérios de testes, sendo eles Switch Cover, Unique Input/Output (UIO) e Distinguishing Sequence (DS), para quatro MEF e avaliar a resposta de cada um, analisando seu custo nos casos de testes gerados.*

1. Introdução

O desenvolvimento de um software conta com diferentes etapas, que podem ser definidas sucintamente como, levantamento e análise de requisitos, implementação, testes e implantação do produto. Todas essas etapas são fundamentais para a qualidade e confiabilidade de funcionamento do software, sendo que cada etapa possui características de desenvolvimento.

A etapa de testes é realizada de modo a determinar se as especificações e funcionamento do sistema estão atuando de forma esperada. Por fim, a implantação do sistema garante a apresentação das funcionalidades e a adaptação para o cenário em que deve ser inserido [Jino et al. 2007].

Há várias abordagens disponíveis para a condução de testes, e é importante possuir um entendimento global do funcionamento do sistema. Dessa forma, é possível garantir a abrangência necessária para abordar todos os possíveis dados e eventos que podem ocorrer durante a execução [Jino et al. 2007].

Idealmente, um programa deveria ser testado abrangendo todas as combinações possíveis de entradas e saídas. Mesmo para programas de menor complexidade, pode ser impossível testar todas as combinações possíveis, podendo haver milhares de cenários de testes. Desse modo, o teste de um sistema complexo levaria muito tempo e exigiria muitos recursos humanos, o que não é viável econômica e operacionalmente [Myers et al. 2004].

A automatização para a geração dos casos de testes tem sido desenvolvida, nos quais diferentes algoritmos abordam várias compreensões de um sistema para geração de casos de teste. Partindo de um modelo que represente o funcionamento do sistema, um critério de geração de casos de teste pode desenvolver um conjunto de testes, que buscam abordar todos os cenários e transições possíveis para um sistema [Hierons and Cengiz Türker 2015].

Utilizando recursos computacionais para compreensão do sistema, um operador humano, apenas, irá focar seus esforços na aplicação dos casos gerados no sistema em funcionamento[Zhu et al. 1997]. Excluindo o operador humano da compreensão do sistema, o desenvolvimento dos casos de testes por meios computacionais atribui ao software maior confiabilidade.

Tendo em consideração a importância de testes de sistemas bem executadas, este estudo visa apresentar e comparar três critérios de testes já definidos em literatura. Os critérios abordados são: Switch Cover [Souza et al. 2017], Unique Input/Output [Sabnani and Dahbura 1988] e Distinguishing Sequence [Gonenc 1970a], identificando qual apresenta melhor desempenho para determinadas MEFs em relação ao custo atrelado as saídas. Será apresentado o conceito de MEFs e dos critérios abordados, os materiais e métodos utilizados para a comparação e as análises realizadas e, por fim, a conclusão desse estudo.

2. Referencial Teórico

Os conceitos e definições descritos, consideram a utilização de máquinas de estados finitos e statecharts na geração de casos de testes para determinado sistema. Sendo que, diferentes algoritmos de geração de casos de testes podem ser usados, possuindo suas características variadas conforme a implementação.

2.1. Máquinas de Estado Finito

As Máquinas de Estados Finito (MEF) são estruturas lógicas que representam um sistema sequencial com um conjunto finito e não vazio de estados. Os exemplos de sistemas que uma MEF pode representar são inúmeros, como programas para a web, automatização robótica, criação mecânica, eletrônica, entre outros [Vieira 2006].

O funcionamento de uma Máquina de Estados consiste na alimentação da máquina a partir de dados de entrada, gerando dados de saída. Ao ocorrer um *evento* de entrada, o estado atual é alterado para um novo estado, correspondente a transição baseada no dado de entrada. A transição para um novo estado pode ser determinada pela entrada recebida ou pela entrada em conjunto com o estado atual [Vieira 2006].

Uma MEF pode ou não ter saídas para uma transição, sendo uma característica importante para diferenciação e aplicação [Rosen 2009]. Outra característica importante é em relação ao comportamento das saídas, existindo dois modelos distintos de MEFs, as de Mealy e as de Moore. Em uma máquina de Mealy as saídas são dependentes do estado atual e da entrada recebida. Já na máquina de Moore apenas o estado atual afeta a saída após a transição ocorrer [Floyd 2015].

[Hopcroft and Ullman 2006] descrevem em detalhes a característica determinística. Definindo como uma MEF determinística quando não é permitido que para um mesmo estado exista múltiplas transições de saída a partir de uma mesma entrada. Já uma MEF não determinística permite que, para uma única entrada, existam várias transições de saída partindo de um mesmo estado.

Além da característica determinística, uma MEF pode ser forte ou fracamente conectada. Se para cada dois estados existir uma sequência de transições que interliguem eles, a MEF é fortemente conectada. Caso contrário, a MEF é fracamente conectada, ou seja, nem para todos os estados é possível chegar a outro [Fantinato 2002]. Uma MEF também pode ser mínima, desse modo deve conter um número mínimo de estados e transições para desempenhar as funções estabelecidas.

2.2. Statecharts

Podemos classificar um sistema reativo como um modelo que interage constantemente com o ambiente, diferindo de um sistema tradicional que fornece um resultado dada a entrada. Tal modelo de sistema é complexo em seu desenvolvimento, assim, [Harel 1987] propôs uma linguagem visual para a especificação e modelagem desses sistemas, denominada *statecharts*.

Os *statecharts* foram desenvolvidos baseados na ideia de aprimorar o modelo clássico de transição de estados [Harel 1987]. Utilizando de artifícios para a execução de paralelismo, hierarquia e comunicação de difusão [Levi 2001], eles permitem uma representação fiel do sistema.

A compreensão de um statechart por um computador, pode ser feita a partir de linguagens de marcação. Desse modo torna-se possível para um software a leitura, alteração e análise de um statechart. Assim, é importante que elementos como estados, eventos e transições sejam obrigatórios para descrição de um statechart.

2.3. Critérios de Geração de Casos de Testes

A aplicação de testes baseado em modelos utiliza de um modelo comportamental, construído usando os requisitos funcionais do software, para gerar casos de testes [Apfelbaum and Doyle 1997]. Essa técnica utiliza das ações e saídas possíveis em sua execução para modelar um software.

A partir da visita de cada ação existente no modelo, um gerador de casos de testes retira sequências de testes para esse sistema [Dalal et al. 1999]. Nas seções a seguir são apresentados os três critérios utilizados nesse trabalho, Switch Cover [Souza et al. 2017], Unique Input/Output [Sabnani and Dahbura 1988] e Distinguishing Sequence [Gonenc 1970a].

2.3.1. Switch Cover

O critério de teste Switch Cover foi proposto em 1976 por Pimot e Rault, e desde então vem sido pesquisado por diferentes grupos, os quais trazem variações do método, sendo uma delas o H-Switch Cover [Souza et al. 2017]. Este critério proporciona uma vasta cobertura da MEF, o qual aplicada, pois define que todas as transições devem ser verificadas [Pimont and Rault 1976].

O critério switch cover utiliza do algoritmo *sequência de De Bruijn* [De Bruijn 1946] para gerar os casos de teste, tento como característica principal a geração de um grafo balanceado a partir de uma MEF. A aplicação do critério pode ser compreendida nas 4 etapas a seguir. Para melhor exemplificação do critério, a Figura 1 ilustra o critério.

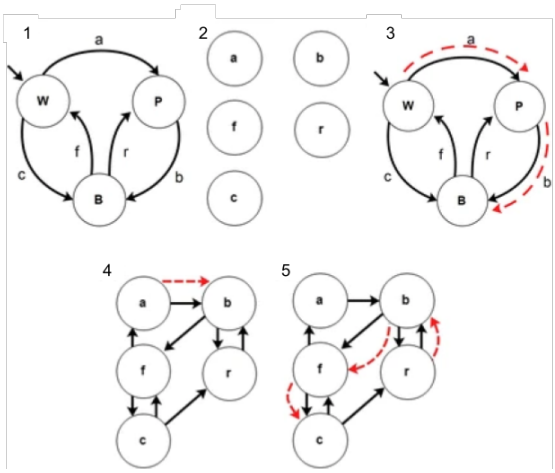


Figura 1. Balanceamento de grafo critério Switch Cover

1. Criação dos vértices para construção do grafo;
2. Criação de novas transições;
3. Balanceamento do grafo;
4. Geração de casos de teste.

O critério de teste switch cover pode deixar em aberto alguns aspectos. O balanceamento do grafo e a geração dos casos de teste, são heurísticas definidas pelo próprio autor que implementa o critério. Assim, podendo ocasionar em uma alteração de desempenho em ambas as etapas[Santiago et al. 2006].

2.3.2. Unique Input/Output

Em 1988, [Sabnani and Dahbura 1988] propuseram uma sequência de identificação de estados, com base em suas entradas e saídas, tendo cada estado da máquina como único. Para a aplicação dessa sequência a MEF deve ser determinística, fortemente conectada e mínima. Essa sequência, denominada *Unique Input/Output sequence (sequência Única de Entrada/Saída)* sendo usada para geração dos casos de teste [Sabnani and Dahbura 1988].

A sequência única de entrada/saída (UES) deve ser diferenciada do critério de teste UIO. A sequência UES distingue um estado ou vértice em uma MEF, e pode ser utilizada para solucionar problemas de verificação de estados. Com a existência de um diagrama de transição de uma MEF, mas sem saber em qual estado está, aplicando uma sequência UES é possível identificar esse estado [Lee and Yannakakis 1996].

O critério de teste UIO utiliza das UES para geração de casos de teste, pode ser aplicado a MEF apresentada na Figura 2, e tem sua implementação dividida em 3 etapas:

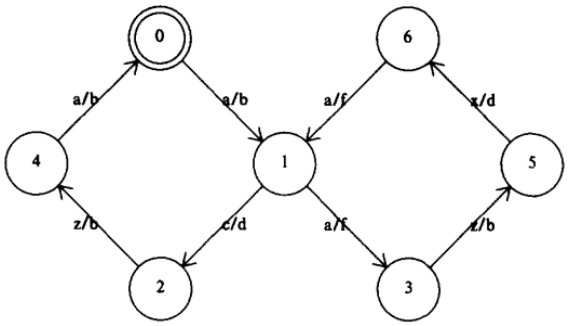


Figura 2. MEF determinística, mínima e fortemente conectada

1. Encontrar a sequência UES para cada vértice;
2. Encontrar subsequência de uma transição;
3. Aplicar a sequência UES para cada vértice.

[Sabnani and Dahbura 1988] ressaltam que, mesmo uma MEF sendo determinística, fortemente conectada e mínima, pode não existir uma UES para determinados vértices, deste modo o critério UIO não pode formar seus casos de teste. Um exemplo onde isso ocorre é caso uma transição possua uma mesma entrada e uma mesma saída, partindo de dois ou mais vértices e indo para um mesmo vértice.

Para exemplificar, a Figura 3 apresenta uma MEF a qual não possui UES para o vértice 1. Como é possível observar, a primeira transição do vértice 1 leva ao vértice 2, porém o vértice 4 possui uma transição com a mesma entrada e saída para o vértice 2. Devido a isso, a UES do vértice 1 não pode começar com a entrada a. Esse mesmo fato ocorre para a outra transição de saída do vértice 1.

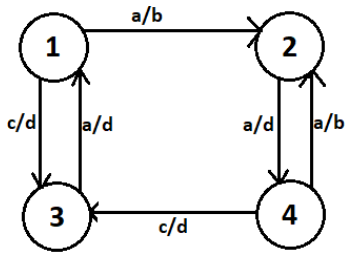


Figura 3. MEF não possui UES

2.3.3. Distinguishing Sequence

Igualmente ao critério UIO, o critério DS para a geração de casos de teste se baseia em uma sequência única de entradas que permitem a identificação de cada vértice em uma MEF [Porto 2013]. Esse método foi originalmente proposto por [Gonenc 1970b] e utiliza uma sequência de distinção (SD), a qual a partir de uma entrada, aplicada a um vértice, gera uma saída distintiva, permitindo determinar qual vértice a MEF se encontra.

Uma MEF deve ser determinística, fortemente conectada e mínima para que o critério seja aplicado [Gonenc 1970b]. Porém, além dessas restrições para certas MEF, a sequência distintiva pode não existir para todos os vértices, limitando o uso do método, o qual não poderá gerar casos de teste [Porto 2013].

O critério DS utiliza para a geração de casos de teste duas sequências concatenadas, a sequência distinção e a subsequência. A sequência inicial manipula a MEF de maneira que, após a execução, o vértice atual seja também o vértice inicial [Seshu 1963]. A subsequência é a mesma utilizada pelo critério UIO, cuja cada transição da MEF seja verificada pelo menos uma vez [Porto 2013].

Para exemplificar e aprofundar na geração de casos de teste pelo critério DS, a seguir são definidas as 3 etapas utilizadas no desenvolvimento:

1. Criação da sequência de distinção;
2. Encontrar subsequência de uma transição;
3. Aplicar a sequência de distinção em todas as subsequências.

A sequência distintiva tem uma similaridade com a UES. Quando uma sequência distintiva existir em uma MEF essa sequência será também uma UES para determinado vértice. Tendo definido esse fato, se uma MEF tem uma sequência distintiva, então todos os vértices possuem uma UES. Quando uma sequência de distinção não existir para determinada MEF o critério de teste DS não pode ser aplicado.

3. Materiais e métodos

A ferramenta WEB-PerformCharts foi utilizada para geração dos casos de testes para cada critério. O processo consiste em uma entrada de um arquivo PcML (linguagem de marcação), contendo o modelo do sistema, selecionado o critério de teste a ser usado e os casos de teste são gerados.

No WEB-Performcharts é possível escolher entre quatro critérios de testes (método T, método DS, método UIO e Switch-Cover) para serem aplicados em um statechart. Por estar disponível na internet auxilia no compartilhamento de casos de teste, permitindo acesso de todos envolvidos no projeto.

Decidir qual critério de teste utilizar para gerar os casos de teste é uma tarefa importante para o projetista. Dado que se deve buscar um critério que melhor se encaixa com a necessidade do sistema. Diversas características dos critérios devem ser analisadas durante a escolha, buscando um critério menor custo sem prejudicar a eficiência dos testes.

Como visto, cada critério de teste tem suas características de execução, a partir disso nem toda MEFs pode ser aplicada, sendo necessário que elas possuam determinadas

Tabela 1. Propriedades de uma MEF para utilização dos critérios Switch Cover, UIO e DS

Propriedades de uma MEF	Switch-Cover	UIO	DS
Minimalidade	X	X	X
Fortemente Conectada	X	X	X
Determinismo	X	X	X
Sequência de Distinção		X	X
UES		X	
Máquina de Mealy		X	X
Possuir saídas		X	X

propriedades. A Tabela 1 apresenta propriedades que devem ser atendidas por uma MEF para a aplicação de cada critério de teste.

A metodologia utilizada para a comparação dos critérios avaliou o custo de cada um para uma mesma MEF. Nesse cenário a palavra custo refere-se a quantidade de casos de teste gerados, e o tamanho de cada caso de teste. A análise de cobertura do statecharts, sendo essa mais complexa, foi desconsiderada para o contexto desse trabalho.

4. Análise e Comparação

O objetivo de comparar os diferentes critérios de testes é analisar o custo de cada um. A primeira comparação entre os critérios utilizou de quatro MEFs, sendo todas aplicadas aos três critérios de testes. Para comparação, todas as MEFs são uma derivação de uma MEF original, a qual é apresentada na Figura 4(a). As demais adaptações dessa MEF são apresentadas na Figuras 4(b), 4(c) e 4(d).

Para cada MEF apenas uma transição foi alterada em relação à original. Comparando a Figura 4(b) com as transições originais, é possível ver que a transição alterada foi que a possui o estado 3 e a entrada *alpha*. Desse modo, as transições de saída do estado 3 são duplicadas, ou seja, tanto para entrada *beta* quanto para *alpha* o estado de destino é 1.

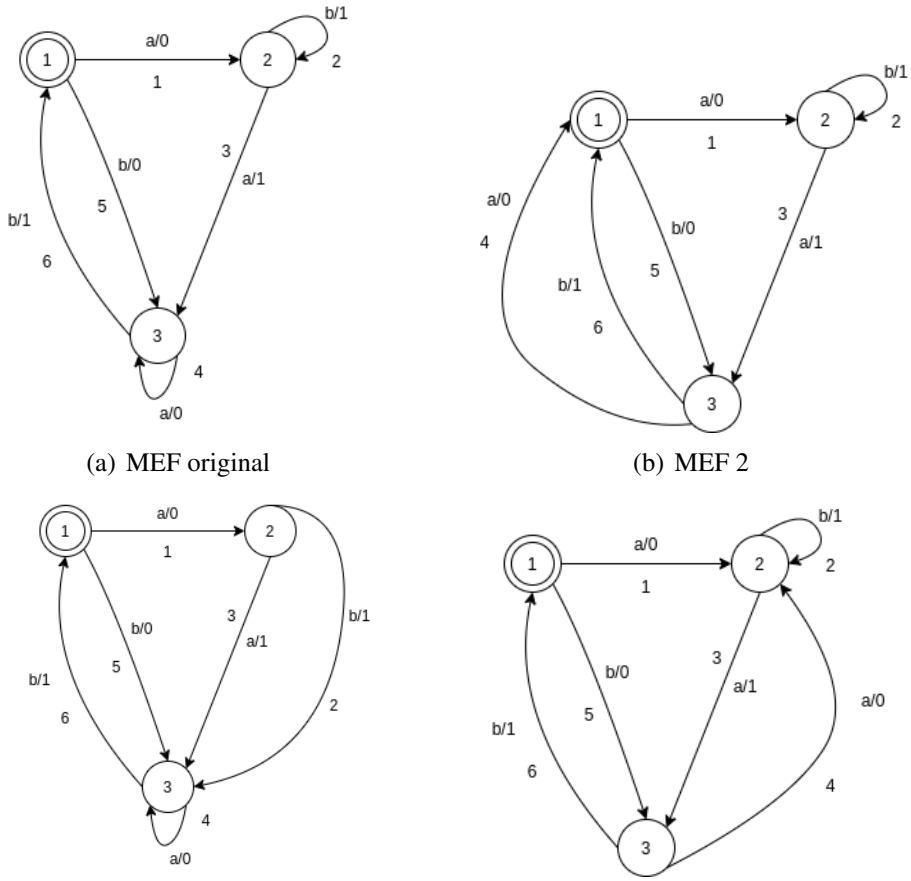
Para a MEF 4(c), as transições são apresentadas na Figura 4(c). A transição alterada possui como estado de origem o 2, e como entrada *beta*. A alteração define o estado de origem como 3, ao fazer isso a transição é duplicada, sendo que para uma entrada *alpha* ela possui o mesmo comportamento.

A última adaptação da MEF original está apresentada na Figura 4(d). Sem duplicar nenhuma transição, a alteração realizada modifica a transição que parte do estado 3 a partir de uma entrada *alpha*.

Tabela 2. Casos de teste

Critérios	Quantidade de casos de teste			
	MEF Original	MEF 2	MEF 3	MEF 4
Switch-Cover	176	16	94	344
UIO	6	5	5	6
DS	6	5	5	6

Analisando o custo e considerando o tamanho dos conjuntos de testes, a Tabela 2 mostra que o número total de casos de teste gerados. Considerando o número de casos de



(a) MEF original (b) MEF 2

(c) MEF 3 (d) MEF 4

Figura 4. MEFs para comparação dos critérios de teste

teste associados a cada critério, tanto o UIO quando o DS fornecem um número menor que o critério switch cover.

Considerando apenas o critério switch cover, e comparando a quantidade de casos de testes obtidas, as MEFs 2 e 3, possuem uma quantidade menor de casos de teste. Nota-se que para essas duas MEFs há duplicidade de transições.

A MEF4 altera a MEF original possibilitando um novo caminho, sem ser esse uma recursividade para o mesmo estado. A partir disso obteve-se uma quantidade maior de casos de teste. Desse modo, pode-se verificar que o critério de teste switch cover é suscetível de grande variação a partir da inserção de novos caminhos.

O custo é definido pela quantidade de casos de testes gerados e pela quantidade de eventos em cada teste. Como intuito de analisar o esforço desempenhado para executar cada caso de teste, a Tabela 3 apresenta alguns valores.

Para cada MEF foi gerada a média e o desvio padrão, da quantidade de eventos que cada caso de teste possui. Ou seja, para a MEF Original, ao aplicar o critério de teste switch cover, cada caso de teste tem em média 8,090 eventos a serem executados.

Analisando a tabela é visto que o critério de teste switch cover possui um número médio de eventos para os casos de teste maior do que os critérios UIO e DS. A diferença

Tabela 3. Média e desvio padrão dos casos de teste

Critérios	Quantidade de casos de teste							
	MEF Original		MEF 2		MEF 3		MEF 4	
	Média	DP	Média	DP	Média	DP	Média	DP
Switch-Cover	8,090	1,834	7,267	3,011	7,667	2,034	10,080	2,834
UIO	2,833	0,408	3	0,707	3,4	0,547	3	0,632
DS	3,667	0,516	2,6	0,547	3,6	0,547	2,667	0,516

entre o desvio padrão também é notável, dado que um valor maior de desvio padrão acarreta maior diferença de tamanho dos casos de teste gerados.

Os valores próximos dos desvios padrão dos critérios UIO e DS sugere que eles mantiveram as quantidades de eventos para caso de testes mais homogêneos. Desse modo pode-se concluir que para as MEFs apresentadas nesse exemplo, o critério que mais necessitaria custos para ser executado é o switch cover. Essa comparação utilizou de MEF similares, mantendo a quantidade de estados, transições, entradas e saídas.

5. Conclusão

Os resultados gerados para os critérios UIO e DS geraram a mesma quantidade de casos de teste, visto que ambos têm seus resultados baseados nas subsequências. O switch cover foi o critério que apresentou um número maior de casos de teste para os quatro modelos de MEFs.

Analisando as variações das MEFs, nota-se que o critério switch cover teve grande variação em relação às quantidades de casos de testes geradas. Para as MEFs onde existia uma duplicação de transição, a quantidade de casos de testes foi menor.

A escolha de um critério de teste para ser aplicada a um sistema deve ser cuidadosa, para que esse percorra de maneira eficiente a MEF e gere quantidades baixas de casos de teste, porém com alta cobertura. A utilização de um critério gerador de casos de teste auxilia no processo, permitindo maior confiabilidade ao sistema de modo geral.

Os critérios gerados foram analisados apenas a partir do custo atrelado as saídas. Outra análise importante a ser executada é a análise de mutação, utilizada para verificar a eficiência e cobertura dos algoritmos.

Referências

Apfelbaum, L. and Doyle, J. (1997). Model based testing.

Dalal, S., Jain, A., Karunanithi, N., Leaton, J., and Lott, C. (1999). Model-based testing in practice. *Proceedings - International Conference on Software Engineering*.

De Bruijn, N. G. (1946). A combinatorial problem. In *Proc. Koninklijke Nederlandse Academie van Wetenschappen*, volume 49, pages 758–764.

Fantinato, M. (2002). Teste de software baseado em máquinas de estados finitos.

Floyd, L. T. (2015). *Digital fundamentals*. Global, Inglaterra, 11 edition.

Gonenc, G. (1970a). A method for the design of fault detection experiments. *IEEE Transactions on Computers*, C-19(6):551–558.

Gonenc, G. (1970b). A method for the design of fault detection experiments. *IEEE Transactions on Computers*, C-19(6):551–558.

Harel, D. (1987). Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274.

Hierons, R. M. and Cengiz Türker, U. (2015). Incomplete distinguishing sequences for finite state machines. *The Computer Journal*, 58(11):3089–3113.

Hopcroft, John E.; Motwani, R. and Ullman, J. D. (2006). *Introduction to automata theory, languages, and computation*. Pearson, Estados Unidos da América.

Jino, M., Maldonado, J. C., and Delamaro, M. E. (2007). *Introdução ao teste de software*. Elsevier - Campus, Brail.

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

Levi, F. (2001). Compositional verification of quantitative properties of statecharts. *Journal of Logic and Computation*, 11(6):829–878.

Myers, G. J., Sandler, C., Badgett, T., and Thomas, T. M. (2004). *The art of software testing*. John Wiley Sons, 2nd ed edition.

Pimont, S. and Rault, J.-C. (1976). A software reliability assessment based on a structural and behavioral analysis of programs. In *Proceedings of the 2nd International Conference on Software Engineering*, ICSE '76, page 486–491, Washington, DC, USA. IEEE Computer Society Press.

Porto, F. R. (2013). Estratégia para geração de sequencias de verificação para máquinas de estados finitos.

Rosen, K. (2009). *Matemática Discreta e suas Aplicações*. Grupo A Educação.

Sabnani, K. and Dahbura, A. (1988). A protocol test generation procedure. *Computer Networks and ISDN Systems*, 15(4):285–297.

Santiago, V., Martins Do Amaral, A. S., Vijaykumar, N. L., Fatima Mattiello-francisco, M. D., Martins, E., and Lopes, O. C. (2006). A practical approach for automated test case generation using statecharts. In *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, volume 2, pages 183–188.

Seshu, S. (1963). Introduction to the theory of finite-state machines. *Proceedings of the IEEE*, 51(9):1275–1275.

Souza, E., Santiago Júnior, V., and Vijaykumar, N. (2017). H-switch cover: a new test criterion to generate test case from finite state machines. *Software Quality Journal*, 25:373–405.

Vieira, N. J. (2006). *Introdução aos fundamentos da computação: linguagens e máquinas*. Cengage Learning, Brasil.

Zhu, H., Hall, P., and May, J. (1997). Software unit test coverage and adequacy. *ACM Computing Surveys - CSUR*, 29(4).