Noname manuscript No.
(will be inserted by the editor)

H-Switch Cover: A new test criterion to generate test case from finite state machines

Érica Ferreira de Souza · Valdivino Alexandre de Santiago Júnior · Nandamudi Lankalapalli Vijaykumar

Received: date / Accepted: date

Abstract Test cases generation based on Finite State Machines (FSMs) has been addressed for quite some time. Model-Based Testing (MBT) has drawn attention from researchers and practitioners as one of the approaches to support software Verification and Validation. Several test criteria have been proposed in the literature to generate test cases based on formal methods, such as FSM. However, there is still a lot to be done on this aspect in order to clearly direct a test designer to choose a test criterion most suitable to generate test cases for a certain application domain. This work presents a new test criterion for model-based test case generation based on FSM, H-Switch Cover. H-Switch Cover relies on the traditional Switch Cover test criterion, but H-Switch Cover uses new heuristics to improve its performance, for example, adoption of rules to optimize graph balancing and traverse the graph for test cases generation. We conducted an investigation of cost and efficiency of this new test criterion by comparing it with Unique Input/Output (UIO) and Distinguishing Sequence (DS). We used two embedded software products (space application software products) and mutation analysis for assessing efficiency. In general, for the case studies proposed in this paper in terms of cost (amount of events) and efficiency (mutation score), H-Switch Cover test criterion presented an average and a standard deviation better than the other two test criteria.

Keywords Software Testing · Model-Based Testing · FSM Test Criteria · H-Switch Cover

Érica Ferreira de Souza
Universidade Tecnológica Federal do Paraná (UTFPR) - Department of Computer
Av. Alberto Carazzai, 1640, 86300-000 Cornélio Procópio, PR, Brazil

Valdivino Alexandre de Santiago Júnior · Nandamudi Lankalapalli Vijaykumar
Instituto Nacional de Pesquisas Espaciais (INPE)
Av. dos Astronautas, 1758, 12227-010 São José dos Campos, SP, Brazil

E-mail: ericasouza@utfpr.edu.br, valdivino.santiago@inpe.br, vijay.nl@inpe.br

1 Introduction

The lack of use of mechanisms for assuring software quality, mainly in critical software, can cause significant losses. Classic examples in which defects in software were the main causes of failure are: a bug in the Intel P5 Pentium floating point unit (FPU), known as "*Pentium FDIV bug*" in that certain floating point division operations performed with these processors produced incorrect results [49]; self-destruction of the rocket Ariane 5, due to a defect in the control software [9]; Therac-25 medical electron accelerator (1985-1987)[39], which caused several losses of human lives; and the Mars Climate Orbiter (1999) [44]. According to Andrade et al. (2013) [1], advances in technology and the emergence of increasingly complex and critical applications require using of improved test strategy, in order to achieve high quality software products.

Software testing is a process/method related to Verification and Validation (V&V) [11,36,43]. Model-Based Testing (MBT) has drawn lot of attention in both industrial and academic areas since it has proved effective by using models to represent system behavior in order to guide the Generation/Selection of Test Cases [20,57,60]. One of the main features of MBT is the automated generation of black box test cases usually based on a formal representation, of the software specification, such as Finite State Machines (FSM) [30,65], Petri Nets [50], Z [34], Vienna Development Model (VDM) [27], and Statecharts [33]. Usually there is a software tool to support modeling as well as the automated test case generation, allowing a test designer to generate complex and large test suites in an automated way, and based on a mathematical formalism.

FSM is a formal modeling technique commonly used for testing due to its rigor and simplicity. FSM has been adopted for modeling reactive systems and protocol implementations for a long time [14,38,54,55]. Reactive systems respond to internal or external stimuli, and it is very suitable to model such systems by means of FSMs because one can basically express the input/output behavior of these applications within the transition labels of an FSM [33]. Embedded software in space applications falls into the category of reactive systems and they have been modeled by means of FSMs [6,14,38,54,55] and Statecharts [60,62]. The published literature shows efforts of the scientific community in analyzing cost-efficiency of test criteria for several techniques like FSMs [16,65] and Statecharts [2,13] aiming to find innovative methods to help test designers to generate test cases in a more systematic manner. Several test criteria[1] are used to generate test cases based on FSM. Some of these are: Switch Cover [53], Transition Tour (TT) [45], Distinguishing Sequence (DS) [31], Unique Input/Output (UIO) [58], Wp [26], State Counting [51] and, recently, H [18], SPY [68], P [67], and Diagnosable Input/Output (DIO) [80].

Switch Cover is an old criterion and it has been investigated for a long time by some research groups. Several versions of this criterion [4,42,53,76] were implemented. According to Arantes et al. (2014) [5], Switch Cover algorithm performs a more refined test coverage where a switch is basically a branch-

[1] In the academic community, a test criterion is also known as a method.

to-branch pair, and test cases consist of every branch-to-branch pair from the graph.

Previous implementations of Switch Cover criterion presented some problems observed over the years: (i) inconsistency between the set of test cases generated and the FSM since we noticed that the test cases did not correspond faithfully to the FSM model; (ii) some implementations of Switch Cover could not properly handle complex FSMs [62]; and (iii) two of the main steps of the Switch Cover criterion are: to obtain a balanced graph and generate a test case based on the Eulerian Graph [12] (explained in Section 3). The original Switch Cover [53] criterion does not provide heuristics[2] for graph balancing and test case generation. This is entirely left to the developers. So, a proper attention must be given to take some decisions regarding these two steps. Usually this has been done in an ad hoc manner without a set of well-defined heuristics. Therefore, the adoption of rules to balancing the graph or traversing the graph without any appropriate heuristics can result in loss of performance due to the computational effort.

According to Endo and Simao (2013) [25], there is a great effort spent in developing new test criteria. These criteria should be able to generate more effective test suites and also reveal as many faults as possible in a particular domain. However, it is common that the proposed new test criteria leads to the necessity to be compared with other traditional criteria. This work presents a new test criterion for FSM, H-Switch Cover, which was derived from Switch Cover. We present our new test criterion and show a cost and efficiency evaluation comparing H-Switch Cover with the traditional criteria, DS and UIO. Results show that H-Switch Cover is more efficient than DS and UIO when some rules are applied so that the algorithm does not generate wrong test cases. Moreover, such rules improve the performance of the criterion and it is now possible to deal with more complex FSMs with several states and transitions. Regarding cost, H-Switch had again a better performance due to the smaller amount of test steps created within the entire test suite.

In order to analyze the test cases efficiency generated by UIO, DS and H-Switch Cover criteria, mutation analysis was adopted. Mutation analysis is used to measure the effectiveness of a test set in terms of its ability to detect faults [37]. Based on mutant analysis, the three criteria analyzed presented good efficiency, but the H-Switch Cover criterion performed slightly better. With respect to cost, we considered the size of the set of test cases generated (number of events). H-Switch Cover presented a better cost, because it generated a smaller amount of test cases. H-Switch Cover criterion also presented, in terms of cost and efficiency, a better average and standard deviation.

This paper is organized as follows. Section 2 presents related work. Section 3 presents in detail the Switch Cover test criterion while Section 4 presents our new test criterion, H-Switch Cover. The study design considering two embedded software products (space application software products) and cost and

[2] We consider that a heuristic is a set of rules that lead to improvement or resolution of problems.

efficiency analysis are shown in Section 5. Analysis of the Results are presented in Section 6. Section 7 presents a general discussion to highlight some points of our research. Finally, Section 8 presents conclusions and future directions for this research.

2 Related Work

FSMs can model a wide variety of systems, such as communication protocols and web applications. Test case generation based on FSMs has been addressed for so long. Several test criteria or test method have been proposed in the literature [74]. This section presents some test criteria for test case generation based on FSM and some areas where they have been investigated.

W test criterion [16] is considered one of the most classic test criteria for test case generation based on FSM. This criterion would be the most efficient to detect the following classes of defects: operation defects, transfer defects, extra state defects and absent state defects. Over the years, several other criteria have emerged, most of them improving this criterion.

The Transition Tour (TT) criterion covers the machine visiting every state and every transition at least once and returns to the initial state [45]. However, the TT criterion obtains only a coverage of transitions. Thus, the criterion does not ensure the detection of transfer defects.

A test criterion proposed by Gonnenc (1970) [31], uses Distinguishing Sequence (DS) to generate test cases. DS is a sequence of input symbols that, when applied to the states of FSM, produces different outputs for each of the states so that it is possible to determine in which state the FSM was originally. However, such a sequence may not exist [30]. It is important to select the smallest DS sequence in order to obtain a smaller set of test cases. According to Gonnenc (1970) [31], the Distinguishing Sequence criterion can only be applied to deterministic, complete, minimal and strongly connected FSM.

The Unique Input/Output test criterion, proposed by Sabnani and Dahbura (1988) [58], produces a state identification sequence called Unique sequence of Input and Output (UIO). UIO is used to verify that whether a given FSM is in a particular state. Thus, for each state of the FSM a distinct UIO sequence must exist, considering the inputs and outputs. Just as in the case with DS, UIO can only be applied to deterministic, complete, minimal and strongly connected FSM. In addition, some improvements to the UIO criterion, Rural Chinese Postman (RCP) [7], MUIO criterion [71], MUIO criterion with overlapping [83], Backward UIO (B-UIO) [70] and circular UIO (C-UIO) [69], were proposed.

The H test criterion is a variation of W where the difference is that it can be applied to partial FSM [41]. It generates a complete test suite and is always applicable for any complete reduced specification.

The Round-Trip Path criterion was proposed by Binder (2005) [11] which also includes an adaptation of W. It uses a strategy that consists of traversing the graph corresponding to the FSM and then generates a tree correspon-

ding to that route. Test cases are derived from sequences of transitions of the generated tree.

The State Counting criterion (SC) [51] reaches the same efficiency to detect defects as W, but it can be applied to FSMs that are partial and not minimal. Most of the criteria require that all states of the FSM are distinguishable. Unlike those methods based on identification of states, the SC method can be applied to deterministic FSM even if their states are not distinguishable.

Several other test criteria can be found in the literature. Table 1 summarizes the main criteria found for FSM.

Table 1: Test Criteria for FSM

Criteria	Year	Description
Switch Cover	1976	All pairs of transitions must be executed [53].
W	1978	It is considered as one of the most classic test criteria for test case generation based on FSM and it is used to distinguish the different states of the specification [16].
TT	1981	The test case starts at the initial state, traverses all transitions at least once and returns to the initial state [45].
DS	1970	Defines a Distinguishing Sequence (DS) that when applied to the states within a FSM produces different outputs for each state and determines at which state the FSM was originally [31].
UIO	1988	Unique input/output sequence for each state of FSM. This enables distinguishing a state from any other [58].
RCP	1988	A Rural Chinese Postman (RCP) tour is used to determine a minimum-cost tour of the transition graph of a finite-state machine. According to Aho et al. (1988) [7], when used in combination with UIO, the technique yields an efficient method for computing a test sequence for protocol conformance testing.
MUIO	1989	Shen et al's method [71] (called MUIO) further improved Aho et al's method for the same class of FSMs by using multiple minimum-length UIO sequences for each state.
MUIO criterion with overlapping	1990	Yang and Ural (1990) [83] describes an optimization method for reducing the length of protocol conformance test sequences by overlapping test subsequences obtained using UIO sequences.

Wp	1991	The Wp method uses the W during the state identification phase while only an appropriate subset is used when checking the ending state of a transition [26].
B-UIO	1991	A Backward UIO (B-UIO) sequence for a state is an input/output behavior which can be observed only if the corresponding state transitions end up at this state. The main idea of this method is to modify the definition of the compound edges in the RCP.
C-UIO	1992	A Circular UIO (C-UIO) sequence for a state is an UIO sequence for this state followed by a B-UIO sequence for the same state. Because the starting and the ending state are the same for a C-UIO sequence, it will go back to the same state after an application to this state.
HSI	1994	In the HSI a family of state identifiers is used for state identification and for transition checking [41].
Round-Trip Path	2001	Traverses FSM's corresponding graph and generates a tree that corresponds to this traverse [11].
State Counting	2005	It has the same efficacy in detecting errors as W, but can be applied to both partial and non-minimal FSM [51].
H	2005	Similar to HSI, H criterion also adopts separating families and can be seen as an improved method. However, H method doesn't use a priori derived state identifiers, but selects state identifiers on-the-fly during the transition testing phase [18].
SPY	2009	SPY is for m-complete test suite generation. The method distributes the sequences of the traversal set over several tests in order to reduce test branching and generates shorter test suites [68].
P	2010	The P method was developed to support incremental testing. The method can be used to generate n-complete test suites in the traditional way. The P method iteratively checks sufficient conditions and applies defined rules to derive the test suite [67].

DIO	2011	Xinchang et al. (2011) [80] proposed a Diagnosable Input/Output (DIO) sequence for identifying a given specified state. Under the assumption that an implementation has no fault or only has IO-correct transfer faults, the DIO sequence can differentiate the associated state from other states.

As seen, several criteria are known for the development of a test suite based on a specification. In the last years research has been focused by research community on the application of these criteria to support different areas and contexts since the use of formal models and specifications can make the testing activity more effective. Some areas are Web Applications [73], generation of improved test cases using Genetic Algorithm [72], Combinatorial Testing [47], Web Services [22,23,82], and studies about the test case suite effectiveness based on Mutation Testing [21]. In our research, we have been investigating the test criteria in the context of embedded software in space applications that fall into the category of reactive systems.

Another aspect investigated is the comparison and improvement of existing test criteria in the literature, context of this work. According to Endo and Simao (2012) [24], the proposal of new criteria motivates the comparison with traditional methods. Souza et al. (2008) [76] presented an empirical evaluation of cost and efficiency among two test criteria of the Statechart Coverage Criteria Family (FCCS) proposed by Souza (2000) [75] all-transitions and all-simple-paths criteria, and the Switch Cover criterion for FSMs. In Simao et al. (2009) [68] there is an approach for comparing criteria. They describe an experimental comparison among different coverage criteria for FSMs, such as state, transition, initialization fault, and transition fault coverage. Dorofeeva et al. (2010) [19], presented an overview and an experimental evaluation of FSM criteria: DS, W, Wp, HSI, UIO, and H. They analyzed the criteria on different aspects, such as test suite length and derivation time. The experiments are conducted on randomly generated specifications and on two realistic protocols.

Fraser and Gargantini (2009) [28] reported on investigations regarding the structure of test suites. They presented a set of experiments performed on the effects of the test case length in a scenario of specification based testing of reactive systems. More recently, Endo and Simao (2013) [25] presented an experimental study that compared traditional and recent FSM-based test generation methods. Endo and Simao compared the test suites generated automatically using the methods W, HSI, SPY, H, and P. They analyzed number of test cases and their length, the total cost (i.e., the length) of each test suite, the effectiveness of the methods using mutation testing for FSMs. In order to conduct the study, FSMs were randomly generated varying numbers of states, inputs, outputs, and transitions.

Considering the criteria as above, one of the most traditional criterion for generating test cases considering FSM is the Switch Cover [53]. The Switch

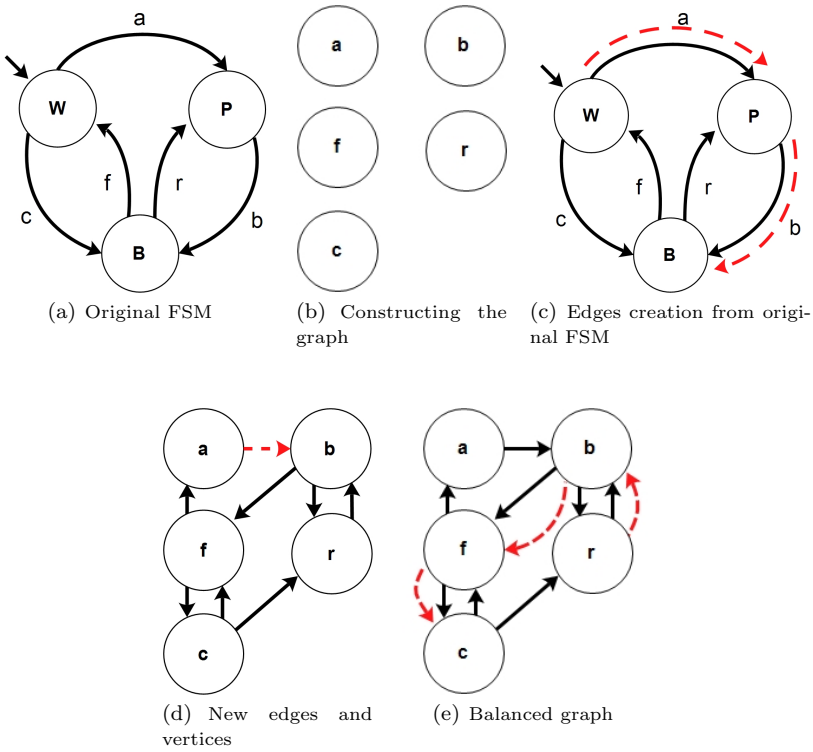
Cover criterion is based on the sequence "*De Bruijn*" [17] used in graph theory where all pairs of adjacent edges must be covered [53]. As we pointed out in Section 1, previous implementations of the Switch Cover criterion presented some inconsistencies between the set of test cases generated and the FSM in that test cases did not correspond faithfully to the FSM model. Another issue was the impossibility to finish creating the test suite. The lack of information and heuristics in the definition of the original Switch Cover for graph balancing and test case generation were the primary reasons for these issues.

According to Burguillo et al. (2002) [15], test case generation at random is not a good strategy. The number of test cases to guarantee an exhaustive coverage may be too large, even infinite. An appropriate strategy should be applied to obtain an efficient test case collection [15]. The main references on Switch Cover presented the algorithm steps, but not its details, e.g., there is no optimization of the algorithm that requires more computational effort in order to improve performance. The previous versions were based on personal criteria constraints, i.e. each author that implemented the Switch Cover criterion, defined his/her own rules to traverse the FSM in order to agree with Switch Cover principles. However, we think that it is necessary to follow a heuristics to guide the implementation of such a criterion, i.e., as highlighted by Burguillo et al. [15], it is necessary to apply an appropriate strategy. Furthermore, we noted that Switch Cover performance could also be improved. This allows reducing the test suite, as well as dealing with complex FSM. Therefore, a new test criterion, H-Switch Cover, has been developed and compared with two traditional criteria, DS and UIO. The development and analysis of the H-Switch Cover criterion are explored in more details in the next sections.

3 The Switch Cover Test Criterion

Before explaining the new test criterion, H-Switch Cover, it is necessary to understand the Switch Cover criterion. Switch Cover is known as "*all combinations*", i.e. all pairs of transitions of an FSM must be covered [53]. The basis for Switch Cover is the "*De Bruijn*" sequence [17]. One of the main characteristics of this criterion is to generate a balanced graph from an FSM. Concepts and algorithms of graph theory can be applied in order to generate test cases, since state-transition diagrams are directed graphs that can be traversed [5]. A graph is an ordered pair $G = (V,E)$ consisting of a set V of vertices (or nodes) and a set E of edges connecting the nodes [12]. From a balanced graph it is possible to generate test cases using a depth-first search approach. All combinations of the balanced graph should be traversed at least once.

Using the FSM shown in Figure 1(a), test cases are generated by Switch Cover as described below:



(a) Original FSM (b) Constructing the graph (c) Edges creation from original FSM

(d) New edges and vertices (e) Balanced graph

Fig. 1 Creation of balanced graph from the original FSM. Source: Adapted from [8].

1. **Create vertices from the original FSM.** The transitions of the FSM must be converted into vertices (or nodes) (Figure 1(b)), in which W is the initial state. So, in this new graph the new vertices "a" and "c" become the initial vertices for the algorithm, because they were the transitions leaving state W.

2. **Add vertices.** Based on the transitions of the original FSM, the graph vertices must be created. For example, in the original FSM, there is a transition a from state W to state P (Figure 1(c)), and there is a transition b leaving state P. Therefore, in the new graph that is being created, an edge is added connecting the new vertices a and b (Figure 1(d)). The same procedure is applied to all other pairs of transitions of the original FSM. Figure 1(d) shows the complete graph. After this process a directed graph is obtained.

3. **Graph balancing.** Now that the graph was constructed, the polarities of the vertices must be balanced by constructing an Eulerian graph (Figure 1(d)). In a Eulerian Graph there is a path where each edge is visited

only once and a graph is Eulerian if there exists a closed trail (Eulerian Tour) containing all edges of E [12]. Balancing is obtained by duplicating the edges in such a way that the number of edges arriving is equal to the number of edges leaving the vertex (i.e. the degree of the vertex is zero). After the graph is balanced, it becomes a multigraph. A multigraph or pseudograph is a graph which is permitted to have multiple directed edges between the same pair of end nodes [10].

4. **Generate Test Cases.** The graph is then traversed generating the test cases always starting at an initial vertex and returning to it. In that example, in the original FSM, the initial state was W: then, in the graph created, when the transitions a and c are transformed into vertices, these become initial vertices in the new graph. The following test cases are generated: *abf, abrbf, cf, crbf.*

We realized that some versions of the algorithm presented in the literature can not support complex FSM and other versions generate test cases that did not reflect exactly the FSM. Not all the transitions of the Eulerian graph are visited, which may generate a test case in which the sequence of test input data/expected results is not consistent with the model. Let us consider the FSM shown in Figure 2. This simple FSM represents a partial model for a simple process scheduler presented in [64]. The model for the process scheduler was developed, in fact, in Statecharts. By using the traditional black box testing technique Equivalence Partitioning [43], where values are assigned to variables and equivalence classes are created, a flat FSM for each proposed equivalence class was created. Thus, Figure 2 shows a machine for one of the equivalence classes.

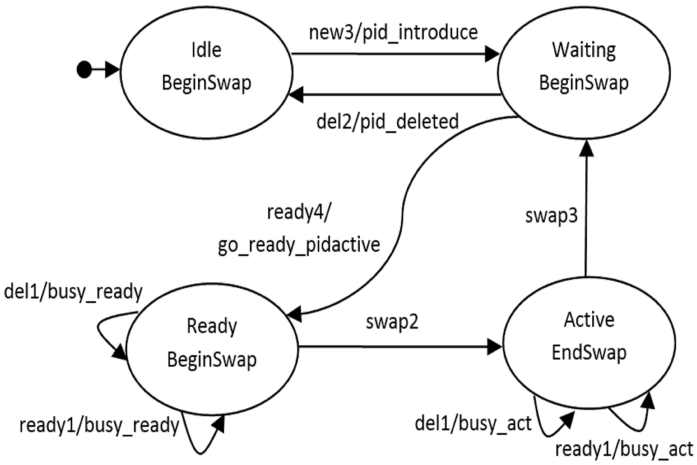


Fig. 2 *An FSM for a simple process scheduler. Source: Adapted from [64]*

The test suite based on the Switch Cover test criterion implemented in the WEB-PerformCharts environment [4–6] is shown below:

1. *new3/pid_introduce, del2/pid_deleted*;

2. *new3/pid_introduce, ready4/go_ready_pidactive, del1/busy_ready, del1/busy_ready, ready1/busy_ready, del1/busy_act, ready1/busy_act, swap3/null, del2/pid_deleted.*

Note that the *swap2/null* input is not present in the second test case. This implies that this test case is not consistent with the model and therefore it can not be executed. In the FSM, the sequence of states of a process is waiting (state *WaitingBeginSwap*), ready (state *ReadyBeginSwap*) to be scaled, and active (state *ActiveEndSwap*). For a process x to change from ready to active, it is necessary that there is a swapping of the current active process, say y, so that x can be the new active process. This swapping is indicated by the *swap2/null* input. After this, another swapping (*swap3/null* input) changes x from active to waiting so that another process, say z, can become active. This is then the problem of this test suite.

In other cases, when the FSM is complex, e.g. the FSM used as case study in this work (Figure 7), not all algorithms from previous versions managed to deal with this FSM. One reason this happens is the algorithm's inability to balance the graph. The graph balancing was unable to be finalized due to lack of appropriate optimization, that is, when a vertex was balanced, other vertices were unbalanced, generating a large amount of duplicated edges to balance the vertex. Furthermore, the problem in the graph balancing, consequently, interfered in the next step of the algorithm: the generation of test cases. Test cases generated became redundant due to the number of generated edges in the graph balancing making the algorithm losing its performance.

4 The H-Switch Cover Test Criterion

In order to solve the aforementioned problems, we developed a new test criterion for FSM test case generation: H-Switch Cover. The fundamental characteristic of H-Switch Cover is the use of the *Hierholzer* [40,46] algorithm adapted to Switch Cover. Other heuristics were defined at specific points in the algorithm to improve its performance and ability to deal with large FSM.

Considering the steps for the algorithm presented above (section 3), the step for creating vertices and edges of the graph from the original FSM are the same as in Switch Cover criterion. Figure 3 shows the algorithm (pseudo-code) to create vertices and edges.

Considering the algorithm from Figure 3, in lines 2-4, in lines 2-4, all of the FSM transitions (T) are converted into a new vertex and included in the graph (corresponding to step 1 of the Switch Cover algorithm - Figure 1(b)). In lines 6-11 new edges are added in the graph (corresponding to step 2 of the

```
   input  : an FSM with a set of states S and a set of transitions T
   output: graph
 1 begin
 2     foreach ti ∈ T do
 3         newVertex ⟵ convertInVertex(ti);
 4         put newVertex in graph;
 5     foreach si ∈ S do
 6         outT ⟵ returnOutGoingTransitionsOf(si);
 7         foreach ti ∈ outT do
 8             sj ⟵ getTargetStateOf(ti);
 9             foreach tj ∈ returnOutGoingTransitionsOf(sj) do
10                 newEdge ⟵ createNewEdge (Label(ti), Label(tj));
11                 put newEdge in graph;
12     return graph;
```

Fig. 3 *Algorithm that converts transitions of original FSM into vertices and adds new edges.*

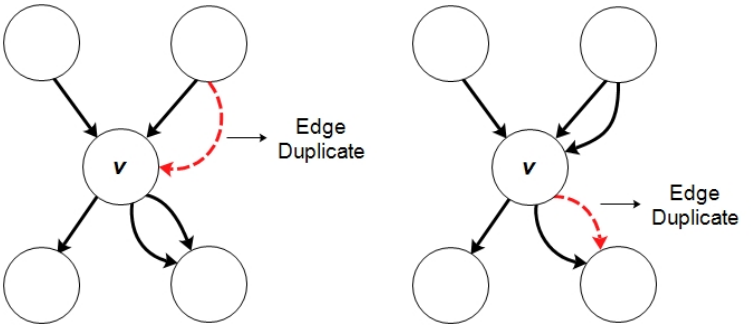
Switch Cover algorithm – Figure 1(c) and 1(d)). For this, first, the states of the FSM are traversed and all outgoing transitions of each state are returned (lines 5-6). The returned transitions are traversed to identify the target states (lines 7-8). For each target state the outgoing transitions (line 9) are identified. This process identifies all pairs of transitions of the original FSM. Finally, considering the pair of transition identified, a new edge is created and added to the graph (lines 10-11)."

The first step of the algorithm has been implemented, it is now necessary to balance the created graph. As highlighted earlier, during the balancing process (step 3 of the original Switch Cover test criterion - Figure 1(e)), several problems occur. For example, whenever a vertex is balanced some other vertex ends up unbalanced. Or, in order to balance a vertex several edges are added, making the generated test case very large or generating an error in the algorithm output. Therefore, in our new test criterion H-Switch Cover, algorithm is show in Figure 5, some rules were created that define the most appropriate position to duplicate (add) a new edge, i.e., the proposed algorithm examines each edge and vertex and returns which edges are more appropriate to be duplicated. Furthermore, these rules improve the algorithm performance. If a vertex is deemed unbalanced, the following rules are applied:

1. **Find the most appropriate initial vertex.** If the indegree of a vertex v is less than its outdegree, a balancing is to be performed. So, a balancing checking is performed with respect to those initial vertices of the directed edges whose terminal vertex is v. This will enable selecting what edges are to be duplicated. However, before performing this, those initial vertices must also be verified with respect to their indegrees, because this may unbalance a balanced vertex. So, a operation is performed in order to solve this, until the best edge to be duplicated without the need to unbalance an

already unbalanced vertex is reached. Figure 4(a) shows a simple example;

2. **Find the most appropriate terminal vertex.** The same procedure described above is applied to the terminal vertex. This is shown in Figure 4(b);



(a) Example: indegree < outdegree (b) Example: indegree > outdegree

Fig. 4 *Added Edge to be duplicated in the graph.*

3. **Analyze balancing process**. If there are vertices that were balanced more times than others, a priority will be given to the vertex that was balanced less number of times;

4. **Analyze edges**. Each vertex also contains a counter with the number of edges it has (indegree and outdegree). If it is necessary to balance a vertex, a priority will be given to a vertex v that is unbalanced. In case all the vertices v have already been balanced, then choose the one with the smallest number of edges.

As previously mentioned, the creation of inappropriate edges may cause inconsistencies in the generated test case, such as extensive sequences and loss of performance. Balancing the graph without any appropriate approach may not end this process when creating new edges. The increase in the number of edges, therefore, will increase the length of test cases. The process to find the most appropriate initial or terminal vertex will help avoiding unbalancing a balanced vertex and it will prevent the creation of inappropriate edges. It is important to make the verification presented above and create the smallest possible number of edges. H-Switch Cover criterion emerged with this objective and consequently reduces the size of test cases. In addition, by analyzing the balancing process and the number of edges (indegree and outdegree), it is possible to control graph balancing since unnecessary edges will not be added without being checked.

The algorithm in Figure 5 shows the main points of the balancing mechanism defined by the H-Switch Cover criterion. As long as there are unbalanced

vertices, the procedure to balance is invoked adding new edges according to the rules implemented and previously presented.

```
   input  : an graph with a set of vertex V
   output: graph balanced
 1 begin
 2     /* A verification is carried out and while graph is unbalanced the balancing
       method is invoked passing a vertex to be balancing */
 3     while verifyBalancing(graph) == false do
 4         foreach vi ∈ V do
 5             if verifyBalancingVertex(vi) == false then
 6                 vertexUnbalanced ⟵ vi;
 7                 balancingVetex(graph, vertexUnbalanced);
 8     balancingVertex(graph, vertexUnbalanced) {
 9     if numIndegree < numOutdegree then
10         newEdge ⟵ returnAppropriateIncomingEdge();
11         createEdge(vertexUnbalanced, newEdge);
12         put newEdge in graph;
13     else
14         newEdge ⟵ returnAppropriateOutgoingEdge();
15         createEdge(vertexUnbalanced, newEdge);
16         put newEdge in graph;
17     }
18     return graph;
```

Fig. 5 *Algorithm for balancing the new graph.*

The algorithm from Figure 5 presents (lines 3-7) a verification which is carried out and while the graph is unbalanced the *balancingVertex* method is invoked passing a vertex to be balanced. In general, the objective *balancingVertex* method is to examine each edge of the vertex and returns which edges are more subject to be duplicated. First, in the *balancingVertex* method, it is verified if the indegree of a vertex is less than its outdegree (lines 9-12). If yes, a balancing is to be performed. The *returnAppropriateIncomingEdge* method (line 10) traverses the edges of the adjacent vertices that need to be balanced and applies the rules to find the more appropriate edges to duplicate and thus balance the vertex (Figure 4(a)). In this method, it is verified if there are vertices that were balanced more times than others; in this case, a priority will be given to the vertex that was balanced less number of times. A counter in the vertex controls the number of times it has been balanced. Each vertex also contains a counter with the number of edges it has (indegree and outdegree). If it is necessary to balance a vertex, a priority will be given to duplicate an edge (coming or outgoing) of a vertex that already is unbalanced. In case all the vertices are balanced, then choose the one with the smallest number of edges. These last two steps allow reducing the creation of many unnecessary

edges. Once the edge to be duplicated is identified, the *edgeCreate* method creates the new edge (line 11) and it is added to the graph (line 12). However, in line 10, if outdegree of a vertex is less than indegree (Figure 4(b)), the *returnAppropriateOutgoingEdge* method is invoked and the same process is conducted, but to the ougoing edge.

The purpose of this graph balancing step is the same as the original algorithm. However, with the implementation of the new approach (rules) to balance the vertices, there was an improvement of the quality of test cases in terms of size, since it optimizes the balancing. As mentioned at the end of Section 2, the main references on Switch Cover presented the algorithm steps, but not their details in its entirety. When there is a complex FSM, it is necessary to follow a common approach to guide the implementation and performance of the criterion.

After the graph balancing, the last step refers to the generation of test case based on the balanced graph (directed multigraph). This step of our algorithm reflects the improvement of the quality of test cases in terms of coverage. H-Switch Cover uses *Hierholzer* heuristic to guarantee that all the edges are visited exactly once (Eulerian path). *Hierholzer* algorithm constructs an Eulerian path suggested from Euler's theorem proof. Euler's theorem says that a connected graph is Eulerian if and only if each vertex (transition arc) has the same degree [40, 46, 12]. Once a graph is balanced, one can apply *Hierholzer* algorithm and generate an Eulerian path. The main steps for implementing the algorithm are:

1. Start with any edge of the initial vertex and select edges not yet visited until a cycle is closed;
2. If there are still any unvisited edges, start with an edge that is a part of an already existing cycle and create a new cycle as in the first step; and
3. If there are no more edges to be visited, an Eulerian cycle must be constructed from the existing cycles, joining them from a common edge.

Using the previous example (Figure 2), the following test case is created from our approach:

Eulerian cycle: *new3/pid_introduced, del2/pid_deleted, new3/pid_introduced, ready4/go_ready_pidactive, del1/busy_ready, ready1/busy_ready, del1/busy_ready, swap2/null, del1/busy_act, ready1/busy_act, del1/busy_act, swap3/null, del2/pid_deleted*

Note that the *swap2* input is present in the test case now, and this implies that this test case is consistent. The algorithm we designed and implemented to traverse the balanced graph and to generate test cases is shown in Figure 6.

In lines 3-13 the algorithm from Figure 6 starts with any edge of the initial vertex and selects some edge not yet visited until a cycle is formed. This cycle is stored in *listCycle*. If there are still any unvisited edges (lines 15-17), the algorithm starts with an edge that is a part of an already existing cycle

```
   input  : graph
   output: testCases
 1 begin
 2     listInitialVertices ← returnInitialVertices(graph);
 3     foreach i ∈ listInitialVertices do
 4         traverseGraph(i);
 5     traverseGraph(i) {
 6     listEdges ← returnListEdges(i);
 7     for j ← 0 to listEdges do
 8         vertexDestination ← returnVertexDestination(j);
 9         if vertexDestination != initialVertex then
10             edge ← returnEdge(vertexDestination);
11             if edge is not visited then
12                 pathVisited ← pathVisited + edge;
13                 traverseGraph(vertexDestination);
14             listCycle ← pathVisited;
15             if edgesNotVisited == true then
16                 newInitialVertex ← returnNewInitialVertex(edgesNotVisited);
17                 traverseGraph(newInitialVertex);
18     }
19     testCases ← createEulerianPath(listCycle);
20     return testCases;
```

Fig. 6 *Algorithm for generation of test cases.*

and creates a new cycle as in the first step. The algorithm will do this until all vertices are visited. Finally, the procedure *createEulerianPath(listCycle)* receives the list of cycles found, and an Eulerian cycle is constructed from the existing cycles, joining them from a common edge.

An important aspect about the 3 criteria, DS, UIO and H-Switch Cover, is with respect to complexity of the algorithms. According to Robinson-Mallett & Liggesmeyer (2006) [56], the generation of a single UIO is of the same complexity as the generation of a DS. Taking into account, that UIO is generated for each state of a machine, time consumption is linear to the number of states. However, the complexities of calculation of UIO increases exponentially $O(n^2)$ with the increase of the number of states and transitions [78]. In the case of Switch Cover or H-Switch Cover criteria, it is important to mention that an Eulerian Cycle (or an Eulerian Path), if exists, the time complexity is linear with respect to the number of edges in the graph ($O(n)$) and it can be solved by the algorithm of Hierholzer [40, 46].

5 Study Design

This section presents the empirical evaluation of cost and efficiency among the three test criteria: UIO, DS and H-Switch Cover. The case studies are two software products embedded into computers of space projects under development

at the *Instituto Nacional de Pesquisas Espaciais* - INPE (National Institute for Space Research).

Twenty one scenarios developed in Statecharts for the case studies were evaluated. The first model refers to the first case study, and the other twenty models refer to the second case study. The second case study considered is a larger and more complex software. It required creating several use cases that addressed the scenarios it contains.

INPE maintains two applications that generate test cases from FSM and Statecharts, that are WEB-PerformCharts [4–6] and *Geração Automática de Casos de Teste Baseada em Statecharts* (GTSC - Automated Test Case Generation based on Statecharts) [57,59]. Both WEB-PerformCharts and GTSC have implemented the test criteria TT[3], UIO, DS and H-Switch Cover. In this work the test cases were generated by GTSC environment where the 3 test criteria are implemented. GTSC allows the generation of test cases automatically from both Statecharts and FSM modeling. When a specification is represented in Statecharts, GTSC converts it into a flat FSM by resolving hierarchy and parallel components. In order to facilitate the interface for the Statecharts specification an XML-based language, PerformCharts Markup Language (PcML) [62] is used. Based on the flat FSM, the test criteria can be selected in order to generate the test cases.

5.1 Case Study I: Alpha, Proton and Electron monitoring eXperiment in the magnetosphere (APEX)

APEX is the software product embedded into an astrophysical experiment computer of a Brazilian space project. The software behavioral model in Statecharts is shown in Figure 7. The software was developed in Java/C/ C++ languages. A proprietary protocol was specified for the communication between the experiment and the On-Board Data Handling (OBDH) computer. OBDH is the satellite platform computer in charge of processing platform and payload information and generating data to be transmitted to Ground Stations. The communication is in primary-secondary mode, where the experiment is totally controlled by OBDH [57].

OBDH and APEX communicate through an RS-232 or USB communication channel. By recognizing a command, the APEX processes and responds to OBDH. The command format message defined in the protocol is composed of 6 (six) fields: *SYNC* (EB9 synchronization value), *EID* (experiment identification), *TYPE* (specifies accepted commands), *SIZE* (amount of Bytes in the DATA field), *DATA* and *CKSUM* (8-bit checksum). SIZE and DATA fields are optional and depend on the type of command.

The model in Figure 7 refers to the main class of the software product embedded into the computer of the scientific instrument: the command recog-

[3] In GTSC, the criterion implemented is the *all-transitions* of the Statechart Coverage Criteria Family (SCCF) proposed by Souza [75].

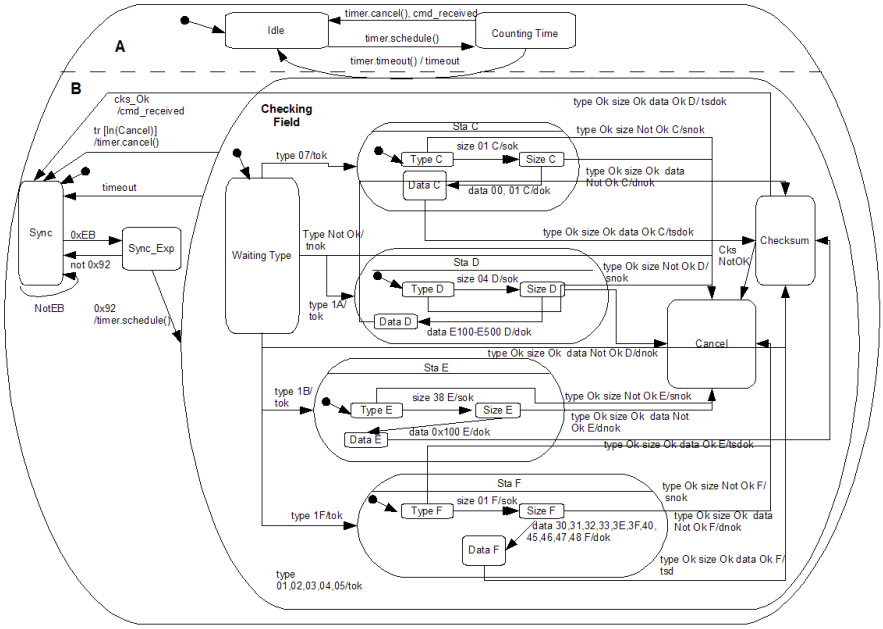


Fig. 7 *Statecharts modeling of the command recognition class of the communication protocol [57]*

nition class. This class is very important because it is responsible for analyzing the conformance of commands received from the OBDH.

5.2 Case Study II: Software for the Payload Data Handling Computer (SWPDC)

The second case study is a software product, called SWPDC, which was developed in the context of the Qualidade do Software Embarcado em Aplicações Espaciais (QSEE - Quality of Space Application Embedded Software) research project [63]. This software has been currently adapted to one on-board computer of a balloon-borne high energy astrophysics experiment under development at INPE. The SWPDC software receives and executes commands from the OBDH, generates housekeeping information, accomplishes data memory management, implements fault tolerance mechanisms, and supports loading of new programs on the fly (firmware update) [66].

As previously mentioned, for the SWPDC case study, 20 (twenty) scenarios were developed for testing and hence 20(twenty) Statechart models were created. For illustration purposes, only the scenario 03 is presented in Figure 8. This scenario refers to exchange of parameters and *housekeeping* data transmission. *Housekeeping* data are composed of several types of data such as temperature, error counters, event reports, among others. Given the large

number of figures and for reasons of space, the other models used can be found in [64].

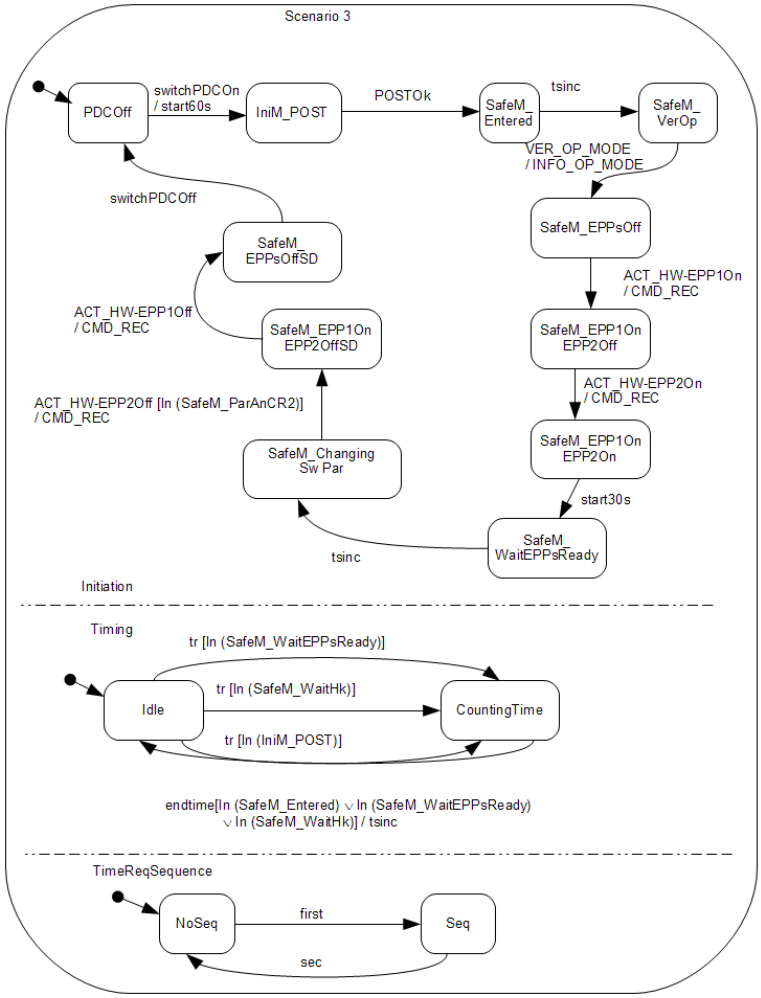


Fig. 8 *Statecharts model scenario 03 (three) SWPDC software [64].*

We accomplished an analysis using measures of dispersion in the SWPDC case study. Our first case study, APEX, was presented in only one Statecharts diagram. On the other hand, in our second case study, SWPDC, given its complexity, 20 (twenty) scenarios were developed for testing and hence 20 (twenty) Statecharts diagrams were created. We analyzed each model separately. However, in order to get a better insight of the results, we presented an analysis of the average and standard deviation of the 20 scenarios.

5.3 Cost and efficiency analysis

Regarding the evaluation of test criteria, measuring the cost and the efficiency of a test set is very challenging. Usually, the size of test suites has been adopted as a measure. The cost is proportional to test suite size that can be measured by counting the number of test cases in a test set [13]. For this work, cost is defined as:

1. The size of test suite. We consider the number of events (steps) as the test case size. In an FSM a transition is a change of state triggered by an input event. An FSM has a state activated always in response to an input event, and this event may cause a change of state, and may produce an output action.

2. Time[4]: This refers to the timed needed by a test suite to kill a mutant. In a real software development environment, test case execution can demand a lot of time due to the size of test suite and the amount of regression testing that can exist in the system testing phase. So, if one test suite can find faults in software earlier than other this can save a significant time during the test execution activity decreasing the cost. It is worth mentioning that in this work, we are not considering the number of test cases in test suite, but the total number of events of all test cases of the test suite.

Efficiency is related to the ability a test suite has to detect a defect. Mutation Analysis can be used for this purpose since mutation analysis is a fault-based testing technique and it can be used to measure the effectiveness of a test set in terms of its ability to detect faults [37]. The mutation score defines a measure of how efficient is of a certain test criterion [48]. It varies from 0 to 1. If a test suite A has a score greater than test suite B, so test suite A is better with respect to efficiency. The mutation score is calculated according to equation 1:

$$MS(P,T) = \frac{DM(P,T)}{M(P) - EM(P)} \tag{1}$$

where:

$MS(P,T)$ = mutation score; $DM(P,T)$ = number of killed mutants; $M(P)$ = total number of mutants; $EM(P)$ = number of equivalent mutants.

In equation 1, the equivalent mutants are those which present the same results as the original program for any test input data. The main problem of the mutation analysis is the huge number of mutants generated. Besides, the computational effort for executing all these mutants becomes very high, as well as the process to identify the equivalent mutants is really hard. Mutation

[4] In the context of this work, the term "time" refers to the instant that a defect was found in the code, that is, the amount of test cases necessary to identify the mutant.

operators selection for this work was based on studies which show code locations where programmers commit greater number of defects. Moreover, it is necessary to identify the features of the language to select operators that are most suitable [48].

6 Analysis of the Results

This section presents the analysis of the results of cost and efficiency of the UIO, DS and H-Switch Cover criteria, for two embedded software, APEX and SWPDC, used as case study.

6.1 Case Study I: APEX

6.1.1 Efficiency analysis

To perform the evaluation in terms of efficiency in the first case study, 19 (nineteen) mutation operators for Java programming language at both method and class-level were used. Based on these 19 (nineteen) mutation operators 202 (two hundred and two) mutants were generated. Table 2 presents operators which were used and the amount of mutants generated for each operator.

Table 2: Mutation Operators applied to APEX software.

Operators	**Description**	**Amount**
ROR	Relational Operator Replacement. Replace relational operators with other relational operators	38
AOIS	Arithmetic Operator Insertion. Insert short-cut arithmetic operators.	20
STRI	Condition Trap of the command *if*	11
JTI	*this* keyword insertion	8
ASRS	Short-Cut Assignment Operator Replacement	3
COR	Conditional Operator Replacement	11
JID	Member variable initialization deletion	6
AMC	Access Modifier Change	5
JSI	Static modifier insertion	2
JSD	Static modifier deletion	6
EOC	Reference comparison and content comparison replac.	4
AOIU	Arithmetic Operator Insertion. Insert basic unary arithmetic operators.	25
COI	Conditional Operator Insertion	8
LOI	Logical Operator Insertion	21
JTD	*this* keyword deletion	4

PRV	Reference assignment with other comparable variable	12
IOP	Overriding method calling position change	3
SSDL	Overriding method deletion	13
JDC	Java-supported default constructor create	2
	TOTAL	202

Each programming language has its own characteristics, which implies the existence of specific faults [48]. Note that to the 202 mutants created from the first case study developed in Java language, ROR, AOIU and AOIS operators showed the larger amount of mutants with 38, 25 and 20 respectively. These mutants are method-level operators and are related to Relational and Arithmetic Operators. Faults that correspond to these types of mutation operators often appear in the code referring to lines or blocks where programmers commit greater number of defects. On the other hand, JDC and JSI operators have had lower number of mutants created. These are mutation operators at the class level and they deal with Java-specific features.

Table 3 shows the results with respect to efficiency of these three test criteria. Mutation score varies from 0 to 1. Closer to 1, more efficient is the test suite due to a certain test criterion. All criteria presented a good score and, therefore, a good efficiency. Although very little, H-Switch Cover and UIO criteria showed a better efficiency than DS. One explanation for this result relies on the fact that if at least one test case of the entire test suite kills a mutant, the entire test suite is said to detect the fault (defect). It does not matter how many test cases can detect the fault as long as one of them is successful.

Table 3: Results with respect to efficiency

Criterion	**Killed**	**Alive**	**Equivalents**	**Mutation Score**
DS	141	61	58	0,98
UIO	136	66	62	0,97
H-Switch Cover	152	50	45	0,98

Among those mutants that are still alive, some may be considered equivalent (see Table 3), that is, although the mutants are syntactically different from the original, if they can exhibit the same behavior, they become equivalent. Evaluating each mutant that remained alive, we identified those considered equivalent mutants. In the first case study, the mutants generated by AMC, JSI, JDI, JSD and PRV operators remained alive. The analysis carried in software code showed that, when using these mutation operators, the changes implemented in the code were not perceived by suites of test cases, but could be considered equivalent.

The mutants that remained alive refer to AOIU and JTI operators. No test case exercised in the code could kill these mutants. Thus, it is necessary that in the specification model, new events that generate test case to kill such mutants must be incorporated.

6.1.2 Cost evaluation

In terms of cost, taking into account the size of test suites, Table 4 shows the total number of events (steps) of the test cases. Considering the number of events associated with all test cases, both the UIO and DS criteria, offer a smaller number of events than H-Switch Cover criterion, leaving it to the test designer choose which situation best fits his/her project.

Table 4: Cost Evaluation - Amount of Events

Criterion	**Amount of Events**
DS	183
UIO	203
H-Switch Cover	722

Another aspect used as measure of cost is the time needed for a test criterion to find a mutant. A test case is considered adequate when at least one test case of the entire test suite kills the mutant. However, to measure what is the cost required to find the mutant, we observe which test case of the test suite was necessary to detect the mutant. Graphics were created to show, by mutation operator, time (or instant) the test suite generated took to find the mutant.

Figures 9 and 10 present cost with respect to time with the Relational Operator Replacement (ROR) operator and Arithmetic Operator Insertion (AOIU) operator, and the instant when a test case finds the mutant in code. Given the large number of mutants, several graphics (23 figures with analyses) were created to show, by mutation operator, time (or instant) the test suite generated to find the mutant. Due to space restrictions, we added only a few of these graphics. The remaining can be found in [77].

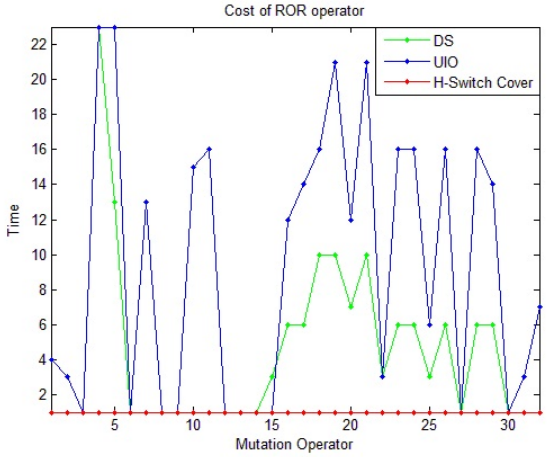


Fig. 9 *Cost - mutation operator ROR*

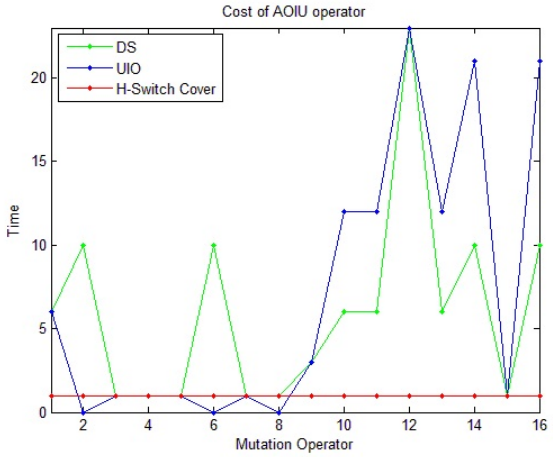


Fig. 10 *Cost - mutation operator AOIU*

Note that H-Switch Cover criterion was faster to find a defect in the code, that is, H-Switch Cover detected a mutant immediately upon the execution of the first test case. This is due to the fact that H-Switch Cover has smaller test suites but its test cases are more complete. The DS and UIO criteria have greater test suites but the test cases are shorter, which detect a mutant, but may not always be at the beginning of the test suite. H-Switch Cover possesses large test case increasing the chances of a defect to be found. We also analyzed only the DS and UIO separately in each figure. Note that DS proved to be

faster than UIO and identified the mutant in less time. Cost of DS is slightly smaller than cost of UIO.

6.2 Case Study II: SWPDC

6.2.1 Efficiency analysis

To evaluate the efficiency of test case generated in the second case study, 09 (nine) mutation operators for C language were used. From these 09 mutation operators, 57 (fifty-seven) mutants for each of the 20 (twenty) scenarios modeled for the SWPDC software were generated. Table 5 presents the operators and the amount of mutants generated for each of the operators.

Table 5: Mutation Operators for the SWPDC software.

Operators	**Description**	**Amount**
ORRN	Change Relational operator	8
SBRC	Break replacement by continue	6
Vsrr	Scalar Variable Reference Replacement	9
SSDL	Statement deletion	7
STRI	Trap on if condition	7
OSSN	Shift assignment mutation	5
SSOM	Sequence Operator Mutation	4
OAAN	Change Arithmetic operator	5
OCNG	Logical context negation	6

Table 6 presents the results related to efficiency of the three test criteria, according to each scenario. The score values of mutation show that all criteria presented good efficiency. In some scenarios, for example, in scenarios 15 (fifteen), 18 (eighteen) and 19 (nineteen), DS and H-Switch Cover were slightly better than UIO in terms of efficiency. And in scenarios 04 (four), 06 (six) and 08 (eight), H-Switch Cover showed better efficiency with respect to UIO and DS. In general, H-Switch Cover criterion was more efficient.

Table 6: Results with respect to efficiency.

Sce.	**DS**	**UIO**	**H-Switch Cover**
01	Killed: 9 Live: 48 Equivalents: 48 Mutation Score: 1	Killed: 9 Live: 48 Equivalents: 48 Mutation Score: 1	Killed: 9 Live: 48 Equivalents: 48 Mutation Score: 1
02	Killed: 15 Live: 42 Equivalents: 42 Mutation Score: 1	Killed: 15 Live: 42 Equivalents: 42 Mutation Score: 1	Killed: 15 Live: 42 Equivalents: 42 Mutation Score: 1
03	Killed: 35 Live: 22 Equivalents: 20 Mutation Score: 0,95	Killed: 35 Live: 22 Equivalents: 20 Mutation Score: 0,95	Killed: 35 Live: 22 Equivalents: 20 Mutation Score: 0,95
04	Killed: 28 Live: 29 Equivalents: 28 Mutation Score: 0,97	Killed: 28 Live: 29 Equivalents: 28 Mutation Score: 0,97	Killed: 29 Live: 28 Equivalents: 28 Mutation Score: 1
05	Killed: 29 Live: 28 Equivalents: 26 Mutation Score: 0,94	Killed: 29 Live: 28 Equivalents: 26 Mutation Score: 0,94	Killed: 29 Live: 28 Equivalents: 26 Mutation Score: 0,94
06	Killed: 29 Live: 28 Equivalents: 25 Mutation Score: 0,91	Killed: 29 Live: 28 Equivalents: 25 Mutation Score: 0,91	Killed: 30 Live: 29 Equivalents: 27 Mutation Score: 1
07	Killed: 35 Live: 22 Equivalents: 19 Mutation Score: 0,92	Killed: 35 Live: 22 Equivalents: 19 Mutation Score: 0,92	Killed: 35 Live: 22 Equivalents: 19 Mutation Score: 0,92
08	Killed: 35 Live: 22 Equivalents: 20 Mutation Score: 0,95	Killed: 35 Live: 22 Equivalents: 20 Mutation Score: 0,95	Killed: 36 Live: 21 Equivalents: 20 Mutation Score: 0,97
09	Killed: 40 Live: 17 Equivalents: 14 Mutation Score: 0,93	Killed: 40 Live: 17 Equivalents: 14 Mutation Score: 0,93	Killed: 41 Live: 16 Equivalents: 13 Mutation Score: 0,93
10	Killed: 36 Live: 21 Equivalents: 17 Mutation Score: 0,90	Killed: 36 Live: 21 Equivalents: 17 Mutation Score: 0,90	Killed: 36 Live: 21 Equivalents: 17 Mutation Score: 0,90
11	Killed: 35 Live: 22	Killed: 35 Live: 22	Killed: 35 Live: 22

	Equivalents: 19 Mutation Score: 0,92	Equivalents: 19 Mutation Score: 0,92	Equivalents: 19 Mutation Score: 0,92
12	Killed: 38 Live: 19 Equivalents: 16 Mutation Score: 0,93	Killed: 38 Live: 19 Equivalents: 16 Mutation Score: 0,93	Killed: 38 Live: 19 Equivalents: 16 Mutation Score: 0,93
13	Killed: 39 Live: 18 Equivalents: 16 Mutation Score: 0,95	Killed: 39 Live: 18 Equivalents: 16 Mutation Score: 0,95	Killed: 39 Live: 18 Equivalents: 16 Mutation Score: 0,95
14	Killed: 40 Live: 17 Equivalents: 16 Mutation Score: 0,98	Killed: 40 Live: 17 Equivalents: 16 Mutation Score: 0,98	Killed: 40 Live: 17 Equivalents: 16 Mutation Score: 0,98
15	Killed: 37 Live: 20 Equivalents: 16 Mutation Score: 0,90	Killed: 37 Live: 20 Equivalents: 16 Mutation Score: 0,90	Killed: 37 Live: 20 Equivalents: 16 Mutation Score: 0,90
16	Killed: 36 Live: 21 Equivalents: 19 Mutation Score: 0,95	Killed: 36 Live: 21 Equivalents: 19 Mutation Score: 0,95	Killed: 36 Live: 21 Equivalents: 19 Mutation Score: 0,95
17	Killed: 37 Live: 20 Equivalents: 18 Mutation Score: 0,95	Killed: 37 Live: 20 Equivalents: 18 Mutation Score: 0,95	Killed: 37 Live: 20 Equivalents: 18 Mutation Score: 0,95
18	Killed: 38 Live: 19 Equivalents: 17 Mutation Score: 0,95	Killed: 37 Live: 20 Equivalents: 16 Mutation Score: 0,90	Killed: 38 Live: 19 Equivalents: 17 Mutation Score: 0,95
19	Killed: 39 Live: 18 Equivalents: 16 Mutation Score: 0,95	Killed: 38 Live: 19 Equivalents: 14 Mutation Score: 0,88	Killed: 39 Live: 18 Equivalents: 16 Mutation Score: 0,95
20	Killed: 37 Live: 20 Equivalents: 17 Mutation Score: 0,93	Killed: 37 Live: 20 Equivalents: 17 Mutation Score: 0,93	Killed: 37 Live: 20 Equivalents: 17 Mutation Score: 0,93

Just as in the first case study, each mutant that remained alive and those that are considered equivalent were evaluated. In most cases, where the mutant remained alive, the test case did not reach the place where the defect was. This is due to the fact that the scenarios were modeled for specific use cases. So, it is necessary that the specification modeled incorporate new events that generate test cases to kill such mutants.

As there are 20 scenarios, an analysis using measures of dispersion may have a better insight into the results of each criterion for efficiency. Table 7 shows the values of average and standard deviation for the Mutation Score of all scenarios.

Table 7: Average and standard deviation for the Mutation Score.

	DS	**UIO**	**H-Switch Cover**
Average	0,9490	0,9425	0,9560
Standard Deviation	0,0271	0,0323	0,0296

We calculated the average of the mutation score for each of the three test criteria. Note that although the score difference is little between the test criteria, the average of mutation score shows that H-Switch Cover is closer to 1, with a score of 0.956, and it can be considered better than the other criteria in terms of efficiency. Now, we want to know the measure of the dispersion of a set of data values from its mean, and so, we use the standard deviation. A high standard deviation indicates that the data points are spread out over a wider range of values, while a low standard deviation (close to 0) indicates that the data points tend to be very close to the mean and it is the expected value of the set. In our analysis, a low standard deviation is better because it means that the efficiency of the test criteria is reasonably uniform regardless of the scenarios. Overall, the three criteria presented a low standard deviation. The UIO criterion has a value greater than the other criteria. The DS and H-Switch Cover criteria show similar dispersions.

6.2.2 Cost evaluation

To evaluate the cost regarding the number of events (steps) pertaining to all generated test cases, in Table 8 the number of events of the test cases for each scenario is shown.

Table 8: Cost Evaluation - Amount of events of the test cases for each scenario.

Scenario	**Amount DS**	**Amount UIO**	**Amount H-Switch Cover**
1	22	20	4
2	77	77	10
3	324	324	23
4	252	252	20
5	989	989	36
6	860	860	39
7	527	495	30
8	434	464	31
9	170	170	16
10	350	377	27

11	434	434	27
12	350	350	27
13	350	350	27
14	350	350	27
15	350	350	27
16	15	15	27
17	434	434	27
18	350	350	27
19	350	350	27
20	350	350	27

Note in Table 8 that the amount of events for DS and UIO criteria is very similar and is greater than H-Switch Cover criterion in all scenarios. This is because DS and UIO criteria create a larger amount of test cases and adding all events related to each test case, increases considerably the cost.

Table 9 shows the average and standard deviation for the amount of events of the 20 scenarios.

Table 9: Average and standard deviation for the amount of events.

	DS	**UIO**	**H-Switch Cover**
Average	366,900	368,150	25,30
Standard Deviation	235,330	234,825	7,935

The average amount of events of the H-Switch Cover is considered better (lower than) in terms of coverage. With respect to the standard deviation, H-Switch Cover shows a much lower dispersion than the other criteria compared with the respective averages, that is, a better result.

Considering the time factor to measure the criteria cost, in each scenario, those test cases that identified a defect first were evaluated. Figure 11 presents the time graph for scenario 03 (three). Graphs generated for the remaining 19 (nineteen) scenarios are in [77]. Evaluating all the generated graphs demonstrated that H-Switch Cover criterion has always presented a better cost in relation to time.

7 Discussion

The application of formal models and specifications can make the testing activity more effective [35]. Model-Based Testing (MBT) has drawn attention from researchers and practitioners since it is an approach to derive test cases from formal models designed to support the test process. Some of the main benefits of MBT are [52, 79, 32]: (i) automatic generation of test cases; (ii) fault detection effectiveness; (iii) reduced time and cost for testing when compared with manual testing; and (iv) improvement of testing quality since a process

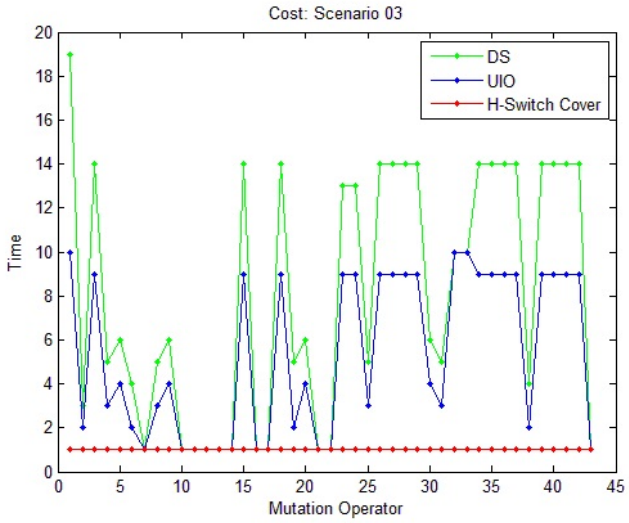


Fig. 11 *Cost for scenarios 03: Time factor.*

based on MBT is more systematic and allows a rigorous, mathematical-based test case generation.

One of the main features of MBT is the automated generation of test cases usually based on a formal representation of the software specification, such as an FSM which we used in this work. FSM has been adopted to generate test cases for different kinds of application, such as Web applications and embedded systems, specially reactive systems [14,3,23,81].

As previously described, Switch Cover is an old criterion and it has been investigated for a long time by some research groups. However, the previous implementations of Switch Cover criterion had some issues observed over the years. Regarding the three issues we mentioned (see Section 1 - Introduction) of previous implementations of the Switch Cover criterion, our new algorithm solved them as follows:

(i) *inconsistency between the set of test cases generated and the FSM.* The first step of the Switch Cover criterion is to create a graph converting the transitions of FSM into vertices of a graph. After the graph is constructed, it is then traversed generating the test cases always starting at an initial vertex and returning to it. However, for some reason previous implementations did not visit all the vertices, generating inconsistency between the set of test cases generated and the FSM since we noticed that the test cases did not faithfully correspond to the modeled FSM. To prevent this problem in this step, H-Switch Cover uses *Hierholzer* heuristic to guarantee that all the edges are visited exactly once (Eulerian path).

(ii) *handling complex FSMs.* We realized that some versions of the algorithm presented in the literature cannot support complex FSM and other versions generate test cases that did not reflect exactly the FSM. To the best of our

knowledge, when the previous switch cover versions were established, tests were performed only on simple FSM and not on real world applications, as is the case of space application software. According to Santiago et al. (2006) [62], in a complex FSM, explosion of test cases frequently takes place and many of the test cases may have sequence of test steps that do not occur in the real implemented software. In this sense, we were careful with the implementation of the new criteria and we prioritize testing with complex models. H-Switch Cover has been in fact employed to validate some critical space application software products developed in our Institute.

(iii) *performance loss in two steps of the switch cover criterion.* In previous versions of Switch Cover we had problems with more complex FSMs, because the graph balancing step was unable to be finalized due to lack of appropriate optimization, that is, when a vertex was balanced, other vertices were unbalanced, generating a large amount of duplicated edges to balance the vertex. The problem in the graph balancing, consequently, interfered in the next step of the algorithm: the generation of test cases. Test cases generated became redundant due to the number of generated edges in the graph balancing making the algorithm losing its performance [29]. To prevent this problem, in our new test criterion H-Switch Cover, some rules were created to define the most appropriate position to duplicate (add) a new edge, i.e, the proposed algorithm examines each edge and vertex and returns which edges are more interesting to be duplicated. Furthermore, our algorithm consequently reduces the size of test cases demanding, in general, less time to execute the entire test suite.

We have adopted mutation to analyze the effectiveness of the test suites generated by H-Switch Cover compared with two traditional criteria, DS and UIO. We accomplished a cost comparison too where we considered cost as the number of events (steps) of the entire test suite. H-Switch Cover criteria had a good efficiency considering its ability to detect faults and a better performance due to the smaller number of events in its test suite.

A limitation of this work is that we did not compare all existing criteria in the literature. We chose the UIO and DS criteria since they are considered traditional and known in the literature by using distinguishing sequences. In addition, there are many criteria for FSM test case generation (see Section 2). Comparing all of them demands the access and/or implementation of such criteria. Thus, it is a tough task to make such a complete comparison. We understand that comparisons like we showed in this research, where complex and real applications are assessed, are valuable towards the improvement of the state of the art.

8 Conclusions

This work presented a new test criterion, called H-Switch Cover. The criterion was integrated into two test environments at INPE (GTSC and WEB-PerformCharts). An empirical investigation of cost and efficiency was per-

formed where H-Switch Cover was compared with two other classical FSM-based test case generation criteria: DS and UIO.

In general, for the case studies proposed in terms of efficiency, the three criteria had good performances with respect to mutation scores related to the two case studies. But DS and H-Switch Cover were slightly better than UIO criterion.

In terms of cost, considering the amount of events, in the first case study UIO and DS criteria were better, but in the second case study, in twenty scenarios evaluated, H-Switch Cover criterion had fewer events, thus with a lower (better) cost.

H-Switch Cover also performed better with respect to the average and standard deviation of the mutation scores and the amount of events of the generated test cases for the 20 scenarios used in the second case study.

With respect to cost (time to detect a defect), H-SwitchCover criterion was faster than the other criteria examined, because it had fewer test cases.

Future directions include evaluation of cost and efficiency of the H-Switch Cover in other application domains and with other test criteria, for instance, test criteria from Statechart Coverage Criteria Family (SCCF). We also intend to use genetic algorithms to improve the performance of some parts of H-Switch Cover. In addition, an empirical study with other professionals shall be made to analyze the impact of the introduction of H-Switch Cover in other settings, in the context of a Model-Based Testing approach.

References

1. Andrade, J., Ares, J., Martínez, M., Pazos, J., Rodríguez, S., Romera, J., Suárez, S. (2013). An architectural model for software testing lesson learned systems. Information and Software Technology, 55, 18-34.
2. Antoniol, G., Briand, L.C., Di Penta, M., Labiche, Y. (2002). A Case Study Using the Round-Trip Path Strategy for State-Based Class Testing. In the 13th International Symposium on Software Reliability Engineering,Annapolis, MD, USA, 269-279.
3. Andrews, A.A., Offutt, J., Alexander, R.T.(2005). Testing web applications by modeling with FSMs, Software and System Modeling, 4(3), 326–345.
4. Arantes, A. O., Vijaykumar, N. L., Santiago, V., Guimaraes, D. (2008). Test Case Generation for Critical Systems through a Collaborative Web-based Tool. In International conference on inovation in software engineering (ISE2008), Viena, Aústria, 163-168.
5. Arantes, A., Santiago Júnior, V. A., Vijaykumar, N. L., Souza, E. F. (2014). Tool support for generating model-based test cases via web. International Journal of Web Engineering and Technology, 9(1), 62-96.
6. Arantes, A. (2008). WEB-PerformCharts: a web-based test case generator from statecharts modeling. Master (Master at Post Graduation Course in Applied Computing) - National Institute for Space Research, São José dos Campos, Brazil.
7. Aho, A. V., Dahbura, A. T., Lee, D., Uyar, M. U. (1988). An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours. In 8th Symposium on Protocol Specification, Testing, andVerification, IFIP, 75-86.
8. Amaral, A. S. M. S. (2005). Geração de casos de testes para sistemas especificados em Statecharts. Master (Master at Post Graduation Course in Applied Computing) - National Institute for Space Research, São José dos Campos, Brazil.
9. European Space Agency (ESA). Ariane-5: Learning from ight 501 and preparing for 502. Paris, France, 1997. ESA Bulletin Nr. 89. Available from: http://www.esa.int/esapub/bulletin/bullet89/dalma89.htm. Accessed March 2015.

10. Bang-Jensen, J., Gutin, G. Z. (2009). Digraphs: Theory, Algorithms and Applications. 2. ed., Springer.
11. Binder R. V. (2005). Testing object-oriented systems models, patterns, and tools. 6. ed., San Diego: Addison Wesley.
12. Bondy, J.A., and Murty, U.S.R. (2008). Graduate Texts in Mathematics: Graph Theory. Editorial Board, USA:Springer.
13. Briand, L.C., Labiche, Y., Wang, Y. (2004). Using Simulation to Empirically Investigate Test Coverage Criteria Based on Statechart. In 26th International Conference on Software Engineering. Edinburgh/Scotland:IEEE, 86-95.
14. Broy, M., Jonsson, B., Katoen, J. P., Leucker, M., and Pretschner, A. (2005). Model-based Testing of Reactive Systems. 1st ed. Springer.
15. Burguillo, J.C., Llamas, M., Fernández, M.J., Robles, T. (2002). Heuristic-driven Techniques for Test Case Selection.Electronic Notes in Theoretical Computer Science, 66(2), 50-65.
16. Chow T. S. (1978). Testing software design modeled by finite-state machines. Transactions on Software Engineering, Piscataway, USA, NJ:IEEE, 4(3), 178-187.
17. De Bruijn, N. G. (1946). A Combinatorial Problem. Koninklijke Nederlandse Akademie v. Wetenschappen 49, 758-764.
18. Dorofeeva, R., El-Fakih, K., Yevtushenko, N. (2005). An improved conformance testing method. In IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE), 204–218.
19. Dorofeeva, R., El-Fakih, K., Maag, S., Cavalli, A.R., Yevtushenko, N. (2010). FSM-based conformance testing methods: a survey annotated with experimental evaluation, Information and Software Technology, 52(12), 1286–1297.
20. El-Far, I. K., and Whittaker, J. A. (2001). Model-based software testing. Marciniak, J. J., editor, Encyclopedia of software engineering. Wiley, USA.
21. El-Fakih, A., A. Simao, Jadoon, N., Maldonado, J. C. (2014). On Studying the Effectiveness of Extended Finite State Machine Based Test Selection Criteria. In International Conference on Software Testing, Verification, and Validation Workshops, Cleveland, OH, 222-229.
22. Endo, A.T., Linschulte, M., Simao, A., Souza, S. R. S. (2010). Event- and Coverage-Based Testing of Web Services. In Fourth International Conference on Secure Software Integration and Reliability Improvement Companion (SSIRI-C), 62-69.
23. Endo, A. T. and Simao, A. (2011). Model-Based Testing of Service-Oriented Applications via State Models. In IEEE International Conference on Services Computing (SCC), 432-439.
24. Endo, A. T. and Simao, A. (2012). Experimental Comparison of Test Case Generation Methods for Finite State Machines. In Fifth International Conference on Software Testing, Verification and Validation. Montreal, QC, 549 - 558.
25. Endo, A. T. and Simao, A. (2013). Evaluating test suite characteristics, cost, and effectiveness of FSM-based testing methods. Information and Software Technology. 55(6), 1045-1062.
26. Fujiwara, S., Bochmann, G. v., Khendek, F., Amalou, M., Ghedamsi, A. (1991). Test selection based on finite state models. Transactions on Software Engineering, IEEE, vol. 17, n. 6, 591-603.
27. Fraser, M. D.; Kumar, K.; Vaishnavi, V. K. (1991). Informal and formal requirements specification languages: bridging the gap. IEEE Transactions on Software Engineering, v. 17, n. 5, 454-466.
28. Fraser, G., Gargantini, A. (2009). Experiments on the test case length in specification based test case generation. In ICSE Workshop on Automation of Software Test (AST), IEEE, 18–26.
29. Fraser, G., Wotawa, A. (2007). Redundancy Based Test-Suite Reduction. Fundamental Approaches to Software Engineering Lecture Notes in Computer Science. Vol. 4422, 291-305.
30. Gill, A. (1962). Introduction to the Theory of Finite-State Machines. New York: McGraw-Hill.
31. Gonnenc, G. (1970). A method for the design of fault detection experiments, IEEE Transactions on computers, 19(6), 551-558.

32. Grieskamp, W., Kicillof N., Stobie, K., Braberman, V. (2011). Model-based quality assurance of protocol documentation: Tools and Methodology. Software Testing, Verification and Reliability, 5(1), 55-71.
33. Harel, D. (1987). Statecharts: a visual formalism for complex systems. Science of Computer Programming. North-Holland. 8, 231-274.
34. Hierons, R. M. (1997). Testing from a Z specification. The Journal of Software Testing, Verification and Reliability, vol. 7, n. 1, 19-33.
35. Hierons, R. M., Bogdanov, K., Bowen, J. P., Cleaveland, R., Derrick, J., Dick, J., Gheorghe, M., Harman, M., Kapoor, K., Krause, P., Luttgen, G., Simons, A. J. H., Vilkomir, S., M. R. Woodward, and Zedan, H. (2009). Using formal specifications to support testing. North-Holland. ACM Computing Surveys (CSUR), 41(2), 1-76.
36. IEEE Std 829. (1998). IEEE standard for software test documentation. New York, NY, USA.
37. Jia, Yue., Harman, M. (2011). An Analysis and Survey of the Development of Mutation Testing. In IEEE Transactions on Software Engineering, 37(5), 649–678.
38. Lee, D. and Yannakakis, M. (2001). Principles and methods of testing finite state machines - a survey. Proceedings of the IEEE, 84, no. 8, 1090–1123.
39. Leveson, N. G. and Turner, C. S. (1993). An investigation of the Therac-25 accidents. Computer. 26, 18-41.
40. Lipschutz, S., Lipson, M. (1997). Matematica discreta. 2 ed. Porto Alegre, RS: Bookman - Coleção Schaum.
41. Luo, G., Petrenko, A. V., Bochmann, G. (1995) Selecting test sequences for partially-specified nondeterministic finite state machines. In International Workshop on Protocol Test Systems, Chapman & Hall, Ltd. London, UK, UK:ACM, 95-110.
42. Martins, E., Sabião, S. B., Ambrosio, A. M. (1999). Condata: a tool for automating specification-based test case generation for communication systems. In International Conference on System Sciences. Hawaii, USA, 303-319.
43. Mathur, A. P. (2008). Foundations of software testing. Delhi, India: Dorling Kindersley (India), Pearson Education in South Asia.
44. NASA jet propulsion laboratory. (2014). Technical report, Cape Canaveral Air Force Station, Florida. Available from: http://www.jpl.nasa.gov/missions/mars-climate-orbiter/. Accessed March 2015.
45. Naito, S., Tsunoyama, M. (1981). Fault detection for sequential machines by transition tours. In Proceedings of the 11th IEEE Fault Tolerant Computing Conference (FTCS 1981), 238-243.
46. Neumann, F. (2004). Expected runtimes of evolutionary algorithms for the Eulerian cycle problem. In Evolutionary Computation, CEC2004, Congress on, 904-910.
47. Nguyen, C. D., Marchetto, A., Tonella, P. (2012). Combining Model-Based and Combinatorial Testing for Effective Test Case Generation. In International Symposium on Software Testing and Analysis (ISSTA), Minneapolis, MN, USA, 15-20.
48. Offutt, A. J. (1993). Experimental Results from an Automatic Test Case Generator. ACM Transactions on Software Engineering Methodology, 2, 109-127.
49. Oishi, M., Tomlin C, Degani, A. (2003). Discrete Abstractions of Hybrid Systems: Verification of Safety and Application to User-Interface Design. NASA/TM-2003-212803. Available from: http://ti.arc.nasa.gov/m/profile/adegani/Discrete%20Abstractions%20of%20Hybrid%20Systems.pdf. Accessed March 2015.
50. Peterson J. L. (1977). Petri nets. ACM Computing Surveys, 9(3), 223-252.
51. Petrenko, A., Yevtushenko, N. (2005). Testing from partial deterministic FSM specifications. Washington, DC, USA. 54(4), 1154-1165.
52. Pretschner, A., Prenninger, W., Wagner, S., Kuhnel, C., Baumgartner, M., Sostawa, B., Zolch, R., Stauner, T. (2005). One Evaluation of ModelBased Testing and its Automation. In 27th International Conference on Software Engineering (ICSE), St Louis, USA, 392-401.
53. Pimont, S., Rault, J. (1976). A software reliability assessment based on a structural and behavioral analysis of programs. In International Conference on Software Engineering (ICSE), CA:IEEE, 486-491.
54. Pinheiro, A. C., Simão, A., and Ambrosio, A. M. (2014). FSM-Based Test Case Generation Methods Applied to Test the Communication Software on Board the ITASAT

University Satellite: A Case Study. Journal of Aerospace Technology and Management, vol. 6, n. 4, 447-461.
55. Pontes, R. P., Véras, P. C., Ambrosio, A. M., Villani, E. (2014). Contributions of model checking and CoFI methodology to the development of space embedded software. Empirical Software Engineering, vol. 19, 39-68.
56. Robinson-Mallett, C., Liggesmeyer P. (2006). State Identification and Verification using a Model Checker, Software Engineering, 131-142.
57. Santiago, V., Vijaykumar, N. L., Guimarães, D., Amaral, A. S., Souza, E. F. (2008). An environment for automated test case generation from Statechart-based and Finite State Machine-based behavioral models. In 4th A-MOST (2008), First IEEE International Conferente on Software Testing Verification and Validation (ICST2008), Lillehammer:IEEE, 63-72.
58. Sabnani, K. K., Dahbura, A. (1988). A protocol test generation procedure, Computer NetWorks and ISDN Systems, 15(4), 285-297.
59. Santiago Júnior, V. A., Vijaykumar, N. L., Souza, E. F., Guimarães, D., Costa, R. C. (2012). GTSC: Automated Model-Based Test Case Generation from Statecharts and Finite State Machines. In Tools Session of the III The Brazilian Conference on Software: Theory and Practice, Natal-RN, 25-30.
60. Santiago Júnior, V. A. (2011). SOLIMVA: A methodology for generating model-based test cases from natural language requirements and detecting incompleteness in software specifications. 264 p. Thesis (Doctorate at Post Graduation Course in Applied Computing) - National Institute for Space Research, São José dos Campos, SP, Brazil.
61. Santiago Júnior, V. A., Vijaykumar, N. L. (2011). Generating model-based test cases from natural language requirements for space application software. Software Quality Journal, vol. 20, 77-143.
62. Santiago, V., Amaral, A. S. M., Vijaykumar, N. L., Matiello-Francisco, M. F., Martins, E., Lopes, O. C. (2006). A practical approach for automated test case generation using Statecharts. In Annual International Computer Software & Applications Conference (COMPSAC) - International Workshop on Testing and Quality Assurance for Component - Based Systems (TQACBS), 30., Chicago, IL, USA, 183-188.
63. Santiago, V., Mattiello, F., Costa, R., Silva, W. P., Ambrósio, A. M. (2007). QSEE project: an experience in outsourcing software development for space. In International conference on software engineering and knowledge engineering (SEKE'07). Boston, USA, 183-188.
64. Santiago Júnior, V. A., Cristiá, M., Vijaykumar, N. L. (2010). Model-based test case generation using Statecharts and Z: a comparison and a combined approach, 72p. (INPE-16677-RPQ/850).
65. 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.
66. Silva, W. P. (2008). QSEE-TAS: Execução automatizada de casos de teste para software embarcado em aplicações espaciais"; São José dos Campos. Master Thesis - National Institute for Space Research (INPE).
67. Simão, A., Petrenko, A. (2010). Fault coverage-driven incremental test generation. Computer Journal, 53, 1508-1522.
68. Simão, A., Petrenko, A., Maldonado, J.C. (2009). Comparing finite state machine test coverage criteria, IET Software, 3(2), 91–105.
69. Shen, X., Li, G. (1992). A new protocol conformance test generation method and experimental results. In ACM/SIGAPP Symposium on Applied computing, New York, NY, USA: ACM, 75-84.
70. Shen, X., Scoggins, S.; Tang, A. (1991). An improved rcp-method for protocol test generation using backward UIO sequences. In Symposium on Applied Computing, ACM, 284-293.
71. Shen, Y. N., Lombardi, F., Dahbura, A. T. (1984). Protocol conformance testing using multiple UIO sequences. In International Symposium on Protocol Specification, Testing and Verification IX, Amsterdam, The Netherlands: North-Holland Publishing Co., 131-143.

72. Shirole, M., Suthar, A., Kumar, R. (2011). Generation of Improved Test Cases from UML State Diagram Using Genetic Algorithm. In 4th India Software Engineering Conference (ISEC), Thiruvananthapuram, Kerala, India, 23-27.
73. Schur, M., Roth, A., Zeller, A. (2014). ProCrawl: Mining Test Models from Multi-user Web Applications. In International Symposium on Software Testing and Analysis (ISSTA), San Jose, CA, 413-416.
74. Soucha, M. (2014). Finite State Machine State Identification Sequences. Open Informatics – Computer and Information Science. Faculty of Electrical Engineering. Department of Cybernetics.
75. Souza, S. R. S. (2000). Validação de especificações de sistemas reativos: definição e análise de critérios de teste. Thesis (PhD in Computational Physics) - Institute of Physics of São Carlos (IFSC/USP).
76. Souza, E. F., Santiago, V., Guimaraes, D., Vijaykumar, N. L. (2008). Evaluation of test criteria for space application software modeling in statecharts. In International conference on inovation in software engineering (ISE 2008). IEEE Computer Society. Viena, Aústria, 157-162.
77. Souza, E. F. (2010). Geração de casos de teste para sistemas da área espacial usando critérios de teste para máquinas de estados finitos. (INPE-16682-TDI/1627). Dissertation (Master in Applied Computing) - National Institute for Space Research (INPE), São José dos Campos, SP, Brazil. Available at: http://urlib.net/8JMKD3MGP7W/36T9LLB. Accessed March 2015.
78. Sun, H., Gaol M., Lian, A. (2001). Study on UIO sequence generation for sequential machine's functional test. In International Conference on, Shanghai, 628-632.
79. Utting, M., Legeard, B. (2006). Practical Model-Based Testing: A tools approach. Waltham, MA, USA. Morgan Kaufmann Publishers Inc.
80. Xinchang Zhang, Meihong Yang, Guanggang Geng, Wanming Luo. (2011). A DFSM-Based Protocol Conformance Testing and Diagnosing Method. Informatica, 22(3), 447-469.
81. Zander, J., Schieferdecker, I., Mosterman, P.(2011). Model-Based Testing for Embedded Systems (Computational Analysis, Synthesis, and Design of Dynamic Systems). New York:Taylor & Francis Group.
82. Wu, C., Huang, C. (2013). The Web Services Composition Testing Based on Extended Finite State Machine and UML Model. In Fifth International Conference on Service Science and Innovation (ICSSI), Kaohsiung, 29-31.
83. Yang, B., Ural, H. (1990). Protocol conformance test generation using multiple UIO sequences with overlapping. SIGCOMM Comput. Commun. Rev., 20(4), 118-125.