Test Case/Step Minimization for Visual Programming Language Models and Its Application to Space Systems

Paulo Nolberto dos Santos Alarcon(✉) and Valdivino Alexandre de Santiago Júnior

Instituto Nacional de Pesquisas Espaciais (INPE),
Av. dos Astronautas, 1758, São José dos Campos, São Paulo, SP, Brazil
paulonsalarcon@gmail.com, valdivino.santiago@inpe.br

Abstract. Visual Programming Languages have been widely used in the context of Model-Based Development, and they find a particular appeal for the design of satellite subsystems, such as the Attitude and Orbit Control Subsystem (AOCS) which is an extremely complex part of a spacecraft. The software testing community has been trying to ensure high quality products with as few defects as possible. Given that exhaustive generation and execution of software test cases are unfeasible in practice, one of the initiatives is to reduce the sets of test cases required to test a Software/System Under Test, while still maintaining the efficiency (ability to find product defects, code coverage). This paper presents a new methodology to generate test cases for Visual Programming Language models, aiming at minimizing the set of test cases/steps but maintaining efficiency. The approach, called specification Patterns, modified Condition/Decision coverage, and formal Verification to support Testing (PCDVT), combines the Modified Decision/Condition Coverage (MC/DC) criterion, Model Checking, specification patterns, and a minimization approach by identifying irreplaceable tests in a single method, taking advantage of the benefits of all these efforts in a unified strategy. Results showed that two instances of PCDVT presented a lower cost (smaller number of test steps) and, basically, the same efficiency (model coverage) if compared with a specialist ad hoc approach. We used the AOCS model of a Brazilian satellite in order to make the comparison between the methods.

Keywords: Model-Based Testing · Test case/step minimization · Model Checking · Specification patterns

1 Introduction

In space systems engineering, quality assurance represents an important role in the system development. Due to the critical and complex nature of a spacecraft, the development of methodologies, methods and techniques to assure its overall quality before launching is highly necessary [1].

O. Gervasi et al. (Eds.): ICCSA 2017, Part VI, LNCS 10409, pp. 160–175, 2017.
DOI: 10.1007/978-3-319-62407-5_11

The Attitude and Orbit Control Subsystem (AOCS) [2] is one of the most important and complex subsystems of a satellite. It is the subsystem responsible for maintaining the spacecraft attitude and orbit ensuring the success of the mission. AOCS must present high quality and robustness. One way to obtain the quality of a system is through well defined processes such as testing. In the context of software development, testing [3,4] is one of the several processes related to Verification & Validation [5].

In order to bring quality assurance to earlier phases, several projects adopt the Model-Driven Development (MDD) [6] strategy where models are tested, verified and improved before the concrete implementation of a complex system/subsystem. As a consequence, models developed can be used to automatically generate test cases through Model-Based Testing (MBT) [7] where such test cases are partially or completely generated from a model describing some aspect (i.e. functionality, performance) of a software product.

A common issue in MBT is test case explosion. In other words, if very detailed models are considered, the amount of test cases to be executed is considerably great [7] making impossible the execution of such test cases withing feasible time. Hence, many studies aim to reduce the amount of test cases necessary to be executed via test case minimization [8–11]. However, test case minimization has been exploited more in the context of regression testing. Thus, it is interesting to investigate and propose new solutions for test cases which are going to be run for the first time.

This work presents a new methodology to generate a minimized set of test cases directly from certain types of Visual Programing Languages (VPLs) while maintaining efficiency of the test suite. Our approach, called specification Patterns, modified Condition/Decision coverage, and formal Verification to support Testing (PCDVT), combines the Modified Decision/Condition Coverage (MC/DC) criterion [12], Model Checking [13], specification patterns mappings for Linear Temporal Logic (LTL) [14], and a minimization approach by identifying irreplaceable tests [11] in a single method, taking advantage of the benefits of all these efforts in a unified strategy. We present an empirical evaluation where we compare two instances of PCDVT against a set of test cases created by a specialist ad hoc approach [15]. All test cases were run over an AOCS model of a Brazilian satellite and the number of test steps and MC/DC coverage obtained for each test suites were compared. The two instances of PCDVT presented a lower cost (smaller number of test steps) and, basically, the same efficiency (model coverage) if compared with the specialist ad hoc approach.

This paper is structured as follows. Section 2 presents the main characteristics the VPLs must possess so that PCDVT can be used. Section 3 presents the PCDVT methodology. Section 4 shows an overview of the Brazilian satellite AOCS model we used as case study. Experimental assessment where we compared two instances of PCDVT with a specialist ad hoc test case generation approach is in Sect. 5. Section 6 presents related work. In Sect. 7, we show the conclusions and future directions of this research.

2 Visual Programing Languages: Required Features

The PCDVT methodology was designed to support block diagram and state-transition VPLs. Our methodology assumes that VPLs must have the following features: (i) the model is composed by blocks and sub-blocks; (ii) each block can present decisions impacting the outcome; and (iii) each possible outcome can lead to different blocks and execution paths. Figure 1 shows an example of VPL model presenting the features described above. Some VPLs which follow these characteristics are SciLab/Xcos [16], Yed [17], and Simulink [18].

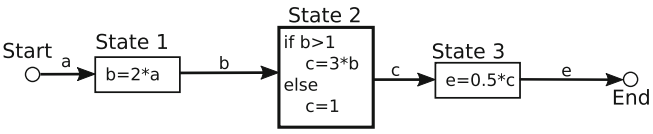


Fig. 1. Example of a general VPL code (model)

3 The PCDVT Methodology

The methodology was developed aiming to make it feasible the generation of test cases directly from some types of VPLs, by reducing the effort to execute the test suite. Furthermore, PCDVT relies on the MC/DC criterion to expedite the qualification of an MDD process since this is an important criterion considered in the DO-178C standard [12]. The PCDVT methodology is presented in Fig. 2. In the next subsections, we will detail the methods and principles of the methodology.

3.1 MC/DC Analysis and Derivation of LTL Properties

MC/DC is a coverage criterion developed in the context of structural (white box) testing aiming to assure that each condition, in a program's decision, must be tested for all possible results at least once, and every program's decision must be tested for all possible results at least once [19].

Figure 3 presents the algorithm to identify the conditions in accordance with MC/DC criterion. In other words, this algorithm takes as input the model and outputs all representations of transitions composed by the source state, s_i, the destination state, s_f, and the condition that independently affects the outcome or True otherwise. Such conditions or True are latter used to generate the LTL properties to support the Model Checking process. In Fig. 3, we have: $M =$ Model; $t =$ State transition in the model; $c_i =$ Condition in a given transition; $I =$ Set of conditions that independently affects the decision; C: Set of all conditions or True values; $s_i =$ Source state, $s_f =$ Destination state; Θ: Set of representations of transitions.

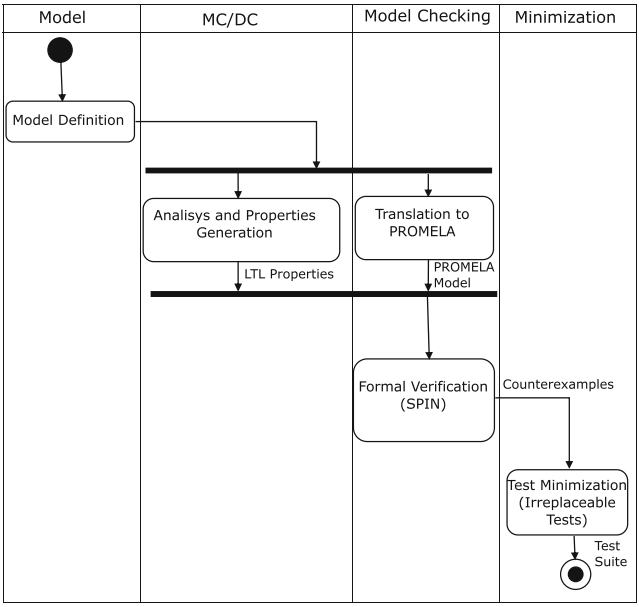


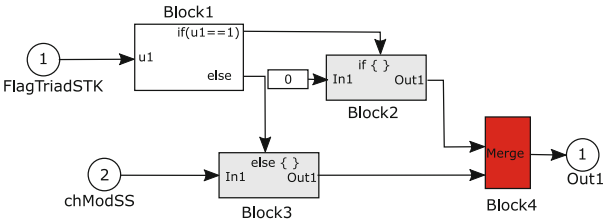
Fig. 2. The PCDVT methodology

Data: Model, M
Result: Set of representations of transitions, Θ
1 $\Theta \leftarrow \emptyset$
2 $C \leftarrow \emptyset$
3 **for** *each* $t \in M$ **do**
4 | **for** *each* $c_i \in t$ **do**
5 | | **if** $c_i \in I$ **then**
6 | | | $C \leftarrow C \cup c_i$
7 | | **end**
8 | | **else**
9 | | | $C \leftarrow C \cup True$
10 | | **end**
11 | **end**
12 **end**
13 $\Theta \leftarrow \Theta \cup \{(s_i, s_f, c_i)\}$
14 **return** Θ

Fig. 3. MC/DC analysis

Figure 4 shows how the algorithm presented works for a simple block. For Block 1, the algorithm will generate 2 conditions to be tested, while for the other blocks it will consider the condition True. Thus, the set of conditions (C) for all blocks, considering the algorithm shown in Fig. 3, becomes $\{u_1 = 1, u_1! = 1, True, True, True\}$.

For each condition obtained via the algorithm in Fig. 3, a LTL property is generated. The LTL properties are derived via two specification patterns for LTL. Considering the Absense Pattern with scope Between Q and R [14], Fig. 5 presents the algorithm, where the source state (s_i) of the transition of the model



Block	Decisions	Conditions	Next Block
1	1	u1=1	Block 2
		u1!=1	Block 3
2	0	True	Block 4
3	0	True	Block 4
4	0	True	Out1

Fig. 4. Example of MC/DC analysis

is considered as Q, and the destination state (s_f) is considered as R. The condition (c) is the formula P of the pattern present. L_a is the set of all LTL properties obtained for each property (l_a) obtained and Θ is the set of representations of transitions obtained previously. Please note the entire pattern is completely negated (!) to force the counterexample generation. In these algorithms, we have the LTL temporal modalities Always ([]); Eventually (<>); Until($\cup$); and Next ($\circ$).

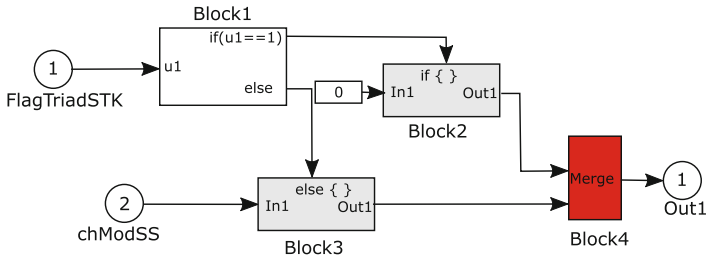
In Fig. 6, we see the Absence Pattern properties derivation for the model in Fig. 4. For each block, at least one LTL property is generated. For decision blocks, each condition identified via MC/DC analysis generates a property for the given transition. When there is no decision, the condition is always set as True and a property is generated.

Data: Set of representations of transitions, Θ
Result: LTL properties due to the Absence Pattern with scope between Q (s_i) and R (s_f)
1 $L_a \leftarrow \emptyset$
2 **for** *each* $c \in \Theta$ **do**
3 $\quad l_a \leftarrow !([]((s_i \& !s_f \& <> s_f) \rightarrow (!c \cup s_f))$
4 **end**
5 $L_a \leftarrow L_a \cup l_a$
6 **return** L_a

Fig. 5. Derivation of Absence Pattern property in LTL

The other specification pattern is the Chain Response with Global scope where the algorithm is presented in Fig. 7. Here, we considered the 2 stimulus-1 response chain, i.e. P responds to S, T. Hence, S is the source state (s_i) of the transition of the model, T is the condition (c) previously identified, and P is the destination state (s_f). L_r is the set of all properties (l_r) obtained and Θ is the set of representations of transitions obtained previously.

Again, the entire pattern is completely negated (!) to force the counterexample generation. An example can be seen in Fig. 8.

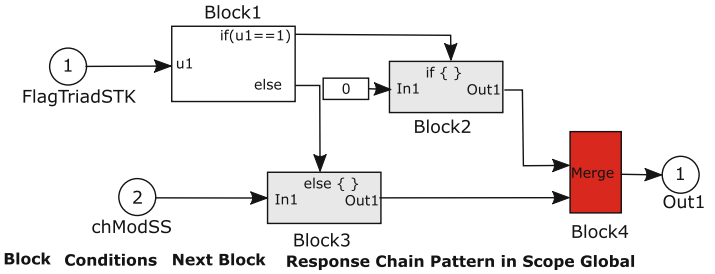


Block	Conditions	Next Block	Absence Pattern in Scope Between Q and R
1	u1=1	Block 2	!([](Block1&!Block2&<> Block2)→(!(u1==1)∪Block2)
	u1!=1	Block 3	!([](Block1&!Block3&<> Block3)→(!(u1!=1)∪Block3)
2	True	Block 4	!([](Block2&!Block4&<> Block4)→(!(True)∪Block4)
3	True	Block 4	!([](Block3&!Block4&<> Block4)→(!(True)∪Block4)
4	True	Out1	!([](Block4&!Out1&<> Out1)→(!(True)∪Out1)

Fig. 6. Example: LTL properties for Absence Pattern

Data: Set of representations of transitions, Θ
Result: LTL Properties due to the Chain Response Pattern with Global scope
$L_r \leftarrow \emptyset$
for *each* $c \in \Theta$ **do**
  $l_r \leftarrow !([](s_i \& \circ <> c \rightarrow \circ(<> (c \& <> s_f))))$
end
$L_r \leftarrow L_r \cup l_r$
return L_r

Fig. 7. Derivation of Chain Response property in LTL



Block	Conditions	Next Block	Response Chain Pattern in Scope Global
1	u1=1	Block 2	!([](Block1&o<> (u1==1)→o(<>((u1==1)&<> Block2)))
	u1!=1	Block 3	!([](Block1&o<> (u1!=1)→o(<>((u1!=1)&<> Block3))))
2	True	Block 4	!([](Block2&o<> (True)→o(<>((True)&<> Block4))))
3	True	Block 4	!([](Block3&o<> (True)→o(<>((True)&<> Block4))))
4	True	Out1	!([](Block4&o<> (True)→o(<>((True)&<> Out1))))

Fig. 8. Example: LTL properties for Response Chain Pattern

3.2 Model Checking

Model Checking is a method used to verify if system behavioral models correspond to the specification [13]. Using a formal specification, typically described as temporal logic, the model checker (tool that has an implementation of Model Checking) explores the model searching for inconsistencies and returning a counterexample showing the flaw.

In addition to the LTL properties, it is necessary to build the formal Transition System (TS) which represents the behavioral modeling of the system under consideration. We used the SPIN Model Checker [20], and thus we need to transform the VPL model into the PROMELA language. We relied on the proposal of [21] where each block diagram is converted into a state-transition model described in PROMELA.

Figure 9 shows an example of this process where first each block identifier is labeled and then the transitions and its trigger conditions are grouped in an intermediate file. With such information, the PROMELA model is built retrieving the information from the original model.

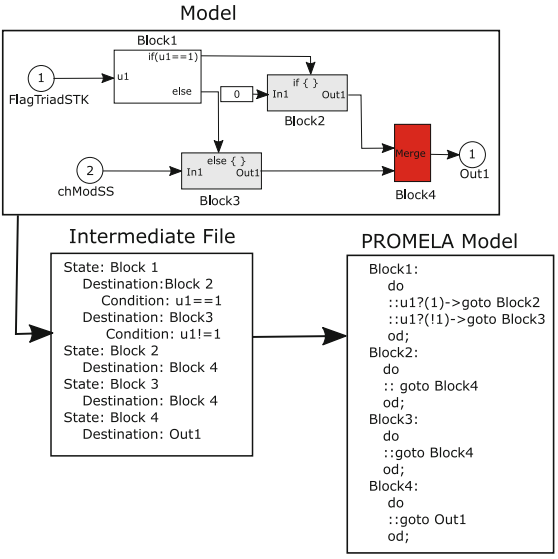


Fig. 9. Example of translation from VPL to PROMELA

The main idea related to Model Checking in PCDVT is to consider the counterexamples as test cases. Each of the properties obtained in the previous steps (Figs. 5 and 7) are checked against the TS.

3.3 Minimization

The last step of PCDVT is the minimization of the test suite previously obtained via Model Checking. Test case minimization is a process of test suite reduction with low impact in the ability to find defects according to this new test suite.

In PCDVT, the minimization process is made via the greedy algorithm combined with the irreplaceable tests evaluation strategy [11]. In this strategy, test cases are evaluated according to their requirement coverage and execution time. Tests with wide requirement coverage and low execution time receive higher weights. When a requirement is verified by only one test case, then it is automatically chosen to compose the optimized test suite. Since the irreplaceable test

strategy was developed in the context of regression tests, it uses information not available during the test creation, such as execution time. The algorithm was slightly adapted in PCDVT. Each LTL property is treated as a requirement and the step count is treated as the execution time, since it directly affects the test duration.

A counterexample analysis was added before the minimization process, were the counterexample steps are analyzed and labeled in accordance with their similarity, based on the work of [10]. Figure 10 describes the redundancy analysis algorithm, where: CE: Set of all counterexamples obtained in the Model Checking step; ce_i: Counterexample in the set CE, being also considered as a set of test steps; η: Labeling operator. An example of redundancy analysis is shown in Fig. 11 where test case 1 is contained in test case 2, then, according with the

Data: Set of all counterexamples, CE
Result: Set of labeled counterexamples, CE_l
$CE_l \leftarrow \emptyset$
for *each* $ce_i \in CE$ **do**
  for *each* $ce_j \in CE, \quad i < j$ **do**
    if $ce_i == ce_j$ **then**
      $ce_i \leftarrow \eta(ce_i, ce_j)$
      $ce_j \leftarrow \eta(ce_j, ce_i)$
    end
    else
      if $ce_i > ce_j \wedge ce_j \subset ce_i$ **then**
        $ce_i \leftarrow \eta(ce_i, ce_j)$
      end
      else
        if $ce_i < ce_j \wedge ce_i \subset ce_j$ **then**
          $ce_j \leftarrow \eta(ce_j, ce_i)$
        end
      end
    end
  end
  $CE_l \leftarrow CE_l \cup ce_i \cup ce_j$
end
return CE_l

Fig. 10. Redundancy analysis algorithm

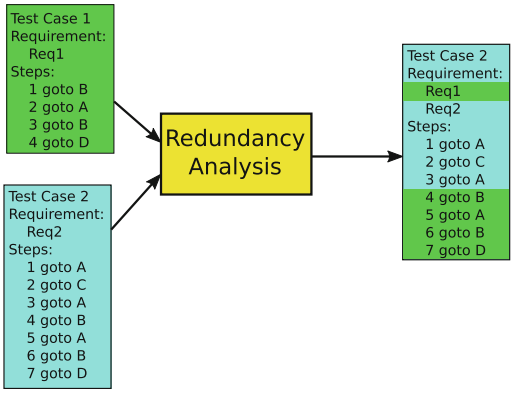


Fig. 11. Example of redundancy analysis

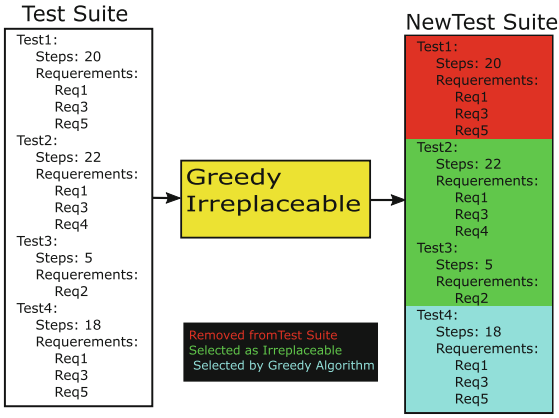


Fig. 12. Example of minimization run

algorithm of Fig. 10, test case 2 is associated with the requirements verified by test case 1. With all requirements mapped for all tests, the Greedy algorithm is executed combined with the irreplaceable tests strategy [11], reducing the test suite (see Fig. 12).

4 Brazilian Satellite AOCS Model

The AOCS [2] is a subsystem that provides information and maintain proper spacecraft attitude and orbit during all phases of the mission, since the separation from the launcher until the end of its operational life. In other words, it is responsible for maintaining the space application position and orientation in space.

The case study is a 2-dimension AOCS model of Lattes-1 satellite [22] presenting 3 operation modes (see Fig. 13), being them Sun Pointing Mode (SPM), Earth Pointing Mode (EPM) and Velocity Control Mode (VCM). Furthermore, it presents a Fault Detection, Isolation and Recovery Module (FDIR), which is able to trigger the SPM. The AOCS was developed to be tolerant to single and combined failure, and the FDIR must trigger the transition to mode SPM when occur composed failure, i.e. two or more components of the same faulted type simultaneously. We selected this case study because it comes from a critical domain (space application) and hence it is a good example to show the usefulness of PCDVT. Also, the use MC/DC criteria to drive the test case generation is important in order to improve the overall code coverage during quality assurance process.

In this model, the following sensors and actuators are considered:

Sun Sensor: Responsible for the sun direction determination. There are 7 sensors, being 3 always illuminated by the Sun independently the satellite orientation; Magnetometer: Responsible for measuring the Earth magnetic field. There

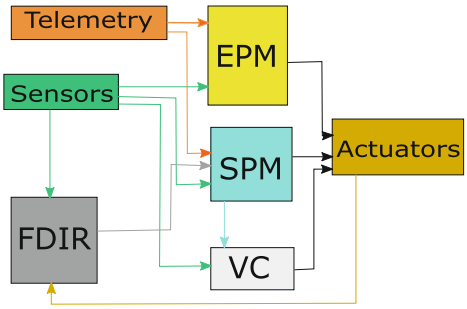


Fig. 13. AOCS block diagram

are 2 magnetometers in the satellite; Gyroscope: Responsible for measuring the satellite angular rate. There are 2 gyroscopes in the satellite; Star Tracker: Sensor responsible for the attitude and angular rate determination with high accuracy. This sensor is used only in the EPM mode and there are two of them; Reaction Wheels: Actuator responsible for the attitude maneuver execution. There are two reaction wheels for redundancy and both are used in the maneuver during normal operation.

4.1 Operation Modes

Sun Pointing Mode (SPM) is the initial mode of AOCS and also considered as contingency mode. It is triggered by telecommands or FDIR (see Fig. 13). It is the simplest mode, where the satellite locks its solar panes and follows the Sun direction. In the mode the mission does not operate and only telemetry and attitude control remain active. The satellite control is done using only sun sensors and magnetometers for attitude determination.

Earth Pointing Mode (EPM) is the mode where the satellite executes its mission, being always triggered by telecommands (see Fig. 13). In this mode all components are active and the satellite keep pointing to the center of Earth by default, but other points can be configured through telecomand.

Velocity Control Mode (VCM) is the mode where the satellite operates when it is in rotation and needs to stabilize (see Fig. 13). In the mode the stabilization process is done in way that no component is damaged. This is a mode that can only be used in the beginning of the mission, since once stabilized, it must keep pointing accordingly to the mission requirements.

5 Experimental Assessment

In order to verify the feasibility of the PCDVT methodology, an experimental evaluation was developed using partial translation of the AOCS model. In this model we took into account the EPM and SPM operation modes, and the FDIR for reaction wheels. Such model is composed by 33 blocks.

Two test suites were generated for the specification patterns previously defined. The test suite generated via the Absence Pattern is called PCDVT-A, and the one generated via the Response Chain pattern is called PCDVT-RC. Results due to PCDVT were confronted against an ad hoc approach created by a domain specialist [15]. Two comparisons were done: cost (amount of test steps) and efficiency (model coverage).

A test case usually comprises several test steps[1]. In other words:

$$tc = \{ts_i \quad | \quad i \in \mathbb{N} \backslash \{0\}\} \tag{1}$$

where $tc =$ test case, and $ts_i =$ test step i. However, one test case, tc_1, might have associated only 1 test step and, for instance, a second test case, tc_2, might be composed of 50 test steps. Thus, comparing the cost of two test suites considering the amount of test cases it is not adequate. Based on this, we defined the cost perspective of our evaluation as the total amount of test steps. Moreover, if we consider a uniform execution time for each test step (i.e. one test step takes α time to be executed), the amount of test steps is directly proportional to the test suite execution time.

In total, 34 LTL properties were generated via PCDVT using both patterns, resulting in 34 test cases in the test suite for each pattern. After the minimization activity, the test suite due to PCDVT-A was reduced to 1 test case while the test

Table 1. Test suite: PCDVT-A

Test case	Test steps	MC/DC coverage
1	25	53%
Total/Mean	25	53%

Table 2. Test suite: PCDVT-RC

Test case	Test steps	MC/DC coverage
1	7	53%
4	25	52%
7	6	53%
11	6	53%
12	10	50%
20	6	57%
23	6	52%
30	7	53%
31	9	59%
Total/Mean	88	53.4%

[1] A test step is an atomic activity to prepare or stimulate the System/Software Under Test. The stimulus can contain the test input data and the expected results.

suite due to PCDVT-RC was reduced to 10 test cases (see Tables 1, 2 and 3[2]). But, as our cost measure is the number of test steps, we presented in Fig. 14 the total test step count due to PCDVT-A, PCDVT-RC, and the specialist

Table 3. Manual test suite [15]

Test case	Test steps	MC/DC coverage
1	3	45%
2	5	59%
3	8	51%
4	8	51%
5	8	52%
6	8	51%
7	8	53%
8	8	52%
9	8	53%
10	8	52%
11	8	53%
12	8	52%
13	8	53%
14	8	53%
15	8	52%
16	8	53%
17	8	52%
18	8	51%
19	8	52%
20	8	53%
21	8	52%
22	8	52%
23	8	53%
24	8	53%
25	8	52%
26	8	53%
27	8	51%
28	8	53%
29	8	53%
30	8	52%
Total/Mean	232	52.23%

[2] Note that, in these tables, the last row implies the total number of test steps and the mean MC/DC coverage.

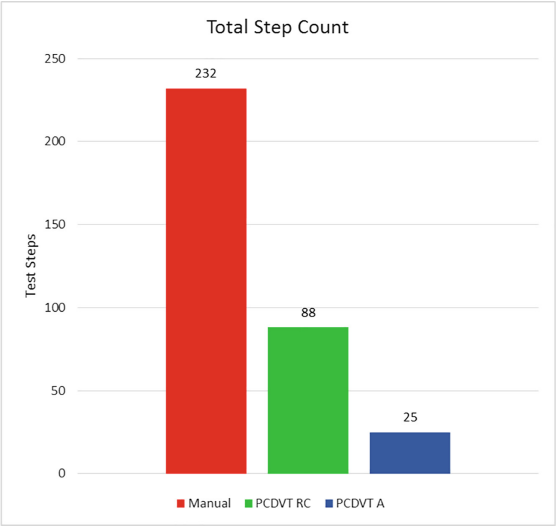


Fig. 14. Comparison of test step count for each test suite

ad doc strategy. As we note, PCDVT-A was the better solution of all with a total of 25 test steps (a reduction of 89% compared to the ad hoc approach). The PCDVT-RC was also better than the specialist approach (82 test steps; reduction of 62%).

Regarding efficiency (model coverage), the three test suites presented similar MC/DC coverage (see Fig. 15). Therefore, the results presented in this section

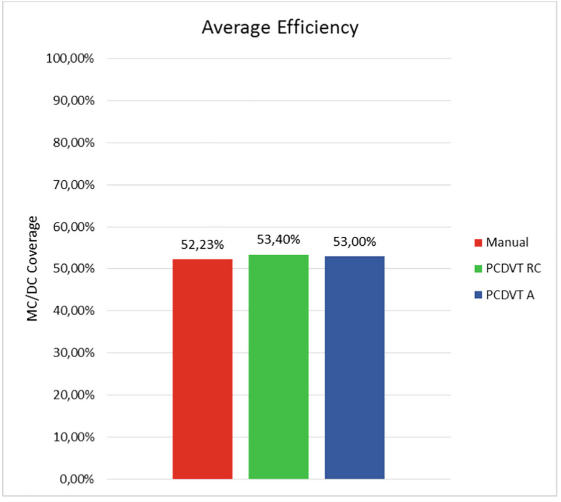


Fig. 15. Comparison of MC/DC coverage for each test suite

show that both PCDVT instances, PCDVT-A and PCDVT-RC, presented a smaller cost (amount of test steps) and basically the same efficiency (model coverage) when compared with a specialist manual and ad-hoc approach [15]. Particularly, the instance PCDVT-A was better than PCDVT-RC in terms of cost and was able to generate test cases for all properties obtained.

6 Related Work

For quality assurance applied to space systems engineering, several techniques have been proposed in areas like test case generation [5,23] and formal verification [24,25]. But no work deals with the use of model checking for test case generation, and does not consider the MC/DC coverage criteria in the space systems engineering context. Most of these MBT techniques are applied to functional test. On the other hand, this work uses these techniques and MC/DC coverage criteria which is associated to structural testing.

Ferrante [9] uses VPL to generate test cases, but Simulink is the only language supported and in this method the models are enriched with test objectives. This technique needs modifications in the models and does not consider coverage criteria. Furthermore, each test case considers only one test objective making the minimization impracticable.

Several approaches have been proposed where Model Checking helps test case generation [9,26–31], but no work considers MC/DC criterion, and only Fraser [26] deals with test case minimization.

Several algorithms and evaluation criteria have been proposed to guide the minimization process, among which stand out Genetic Algorithms [32], code coverage [10,26], requirement coverage [8,11], defects previously found [33], and test execution time [11]. However most of the techniques were developed to be applied in the context of regression testing by using information obtained after a history of executions [11,33,34]. But, such information are not available during the test case generation for the first time.

7 Conclusions

In this paper, we presented a new methodology, PCDVT, in order to provide a minimized test suite (regarding the number of test steps) for VPL models. PCDVT combines several concepts related to software testing (MC/DC, test minimization) and formal verification (Model Checking, specification patterns) to generate a set of test cases that in the end aims to derive a reduced set of test steps, demanding less effort to be executed.

Results have shown that 2 instances of PCDVT, PCDVT-A and PCDVT-RC, were better in terms of cost, and presented basically the same efficiency of an specialist ad-hoc approach. Case study was a non-trivial system, the AOCS of a Brazilian satellite. Particularly, the Absence Pattern with scope between Q and R was the best of all three approaches. This demonstrates the potential of this strategy for complex projects of space systems and other VPL applications.

The PCDVT methodology is partially automated. Hence, in the future we aim to complete the full automation of the tool that supports the methodology. We also intend to consider a more thorough model of the AOCS, with all sensors and actuators. Considering this expanded model, we will compare PCDVT with the specialist approach in terms of the cost and efficiency as defined in this research. This comparison will be done via a rigorous evaluation, e.g. a controlled experiment or quasiexperiment.

References

1. Pisacane, V.L.: Fundamentals of Space Systems. Oxford University Press, New York (2005)
2. Wertz, J.R., Larson, W.J.: Space Mission Analysis and Design. Microcosm Press, Hawthorne (1999). 976 p.
3. Mathur, A.P.: Foundations of Software Testing. Dorling Kindersley (India), Pearson Education in South Asia, Delhi (2008). 689 p
4. Delamaro, M.E., Maldonado, J.C., Jino, M.: Introdução ao teste de Software. Elsevier, Brasil (2007)
5. Santiago Júnior, V.A.: SOLIMVA: a methodology for generating model-based test cases from natural language requirements and detecting incompleteness in software specifications. Ph.D. thesis, Instituto Nacional de Pesquisas Espaciais (INPE) (2011)
6. Balasubramanian, K., Gokhale, A., Karsai, G., Sztipanovits, J., Neema, S.: Developing applications using model-driven design environments. Computer **39**(2), 33–40 (2006)
7. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing approaches. Softw. Test. Verif. Reliab. **22**(5), 297–312 (2012)
8. Campos, J., Abreu, R.: Encoding test requirements as constraints for test suite minimization. In: 2013 Tenth International Conference on Information Technology: New Generations (ITNG), pp. 317–322. IEEE (2013)
9. Ferrante, O., Ferrari, A., Marazza, M.: Model based generation of high coverage test suites for embedded systems. In: 19th IEEE European Test Symposium, vol. 99(2), pp. 335–337 (2014)
10. Fraser, G., Wotawa, F.: Mutant minimization for model-checker based test-case generation. In: Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION, TAICPART-MUTATION 2007, pp. 161–168. IEEE (2007)
11. Lin, C., Tang, K., Kapfhammer, G.M.: Test suite reduction methods that decrease regression testing costs by identifying irreplaceable tests. Inf. Softw. Technol. **56**, 1322–1344 (2014)
12. Holloway, C.M.: Towards understanding the do-178c/ed-12c assurance case. In: 7th IET International Conference on System Safety, Incorporating the Cyber Security Conference 2012, pp. 1–6 (2012)
13. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)
14. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Property specification patterns for finite-state verification. In: Proceedings of the Second Workshop on Formal Methods in Software Practice, pp. 7–15. ACM (1998)
15. Alarcon, P.N.S.: Test specification and procedures for AOCS. Technical report, São José dos Campos (2013)

16. Janík, Z., Žáková, K.: Online design of Matlab/Simulink and SciLab/Xcos block schemes. In: 2011 14th International Conference on Interactive Collaborative Learning (ICL), pp. 241–247 (2011)
17. Wiese, R., Eiglsperger, M., Kaufmann, M.: yFiles visualization and automatic layout of graphs. In: Jünger, M., Mutzel, P. (eds.) Graph Drawing Software, pp. 173–191. Springer, Heidelberg (2004). doi:10.1007/978-3-642-18638-7_8
18. Hanselman, D.C., Littlefield, B.: Mastering Matlab 7. Pearson/Prentice Hall, Upper Saddle River (2005)
19. Chilenski, J.J., Miller, S.P.: Applicability of modified condition/decision coverage to software testing. Softw. Eng. J. **9**(5), 193–200 (1994)
20. Bartocci, E., DeFrancisco, R., Smolka, S.A.: Towards a GPGPU-parallel SPIN model checker. In: Proceedings of the 2014 International SPIN Symposium on Model Checking of Software, pp. 87–96 (2014)
21. Yamada, C., Miller, D.M.: Using spin to check simulink stateflow models. Int. J. Netw. Distrib. Comput. **4**(1), 65–74 (2016)
22. Araujo, H.A.B.: AOCS design specification. Technical report, São José dos Campos (2012)
23. Alarcon, P.N.S., Carvalho, F.G.M., Simoes, A.R.: Geração automática de casos de teste aplicada ao projeto de aocs de satélites artificiais. In: XX Congresso Brasileiro de Automática (CBA 2014), pp. 1652–1659 (2014)
24. Gan, X., Dubrovin, J., Heljanko, K.: A symbolic model checking approach to verifying satellite onboard software. Sci. Comput. Program. **82**, 44–55 (2014)
25. Nardone, V., Santone, A., Tipaldi, M., Glielmo, L.: Probabilistic model checking applied to autonomous spacecraft reconfiguration. In: 2016 IEEE Metrology for Aerospace (MetroAeroSpace), pp. 556–560 (2016)
26. Fraser, G., Wotawa, F., Ammann, P.E.: Testing with model checkers: a survey. Softw. Test. Verif. Reliab. **19**(3), 215–261 (2009)
27. Ferrante, O., Marazza, M., Ferrari, A.: Formal specs verifier ATG: a tool for model-based generation of high coverage test suites. In: 19th IEEE European Test Symposium, vol. 99(2), pp. 335–337 (2014)
28. Zeng, H., Miao, H., Liu, J.: Specification-based test generation and optimization using model checking. In: Symposium on Theoretical Aspects of Software Engineering (TASE 2007) (2007)
29. Enoiu, E.P., Causevic, A., Ostrand, T.J., Weyuker, E.J., Sundmark, D., Pettersson, P.: Automated test generation using model checking: an industrial evaluation. Int. J. Softw. Tools Technol. Transf. **18**, 335–353 (2014)
30. Yeolekar, A., Unadkat, D., Agarwal, V., Kumar, S., Venkatesh, R.: Scaling model checking for test generation using dynamic inference. In: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, pp. 184–191 (2013)
31. Gent, K., Hsiao, M.S.: Functional test generation at the RTL using swarm intelligence and bounded model checking. In: 2013 22nd Asian Test Symposium, pp. 233–238 (2013)
32. Wang, S., Ali, S., Gotlieb, A.: Minimizing test suites in software product lines using weight-based genetic algorithms. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, pp. 1493–1500 (2013)
33. Dandan, G., Tiantian, W., Xiaohong, S., Peijun, M.: A test-suite reduction approach to improving fault-localization effectiveness. Comput. Lang. Syst. Struct. **39**(3), 95–108 (2013)
34. Singh, R., Santosh, M.: Test case minimization techniques: a review. Int. J. Eng. Res. Technol. (IJERT) **2**(12) (2013)