Automated test case generation from high-level logic requirements using model transformation techniques

Oyindamola Olajubu
Division of Computing
University of Northampton, UK
oyindamola.olajubu@northampton.ac.uk

Suraj Ajit
Division of Computing
University of Northampton, UK
suraj.ajit@northampton.ac.uk

Mark Johnson
Division of Computing
University of Northampton, UK
mark.johnson@northampton.ac.uk

Scott Turner
Division of Computing
University of Northampton, UK
scott.turner@northampton.ac.uk

Scott Thomson
GE Aviation,
Cheltenham, UK
scott1.thomson@ge.com

Mark Edwards
GE Aviation,
Cheltenham, UK
mark.edwards4@ge.com

***Abstract*—It is not uncommon for industries to use natural language to represent high-level software requirement specifications. It is also not uncommon for these requirement specifications to be translated into design and used further for implementation and generation of test cases in the software engineering lifecycle. These requirements are often ambiguous, incorrect, and incomplete. Finding them late in the development lifecycle proves very expensive and lowers the productivity. This paper reports on the experience of applying model-based technologies from academia to a real-world problem domain in the aviation industry to improve the productivity. The paper focuses on the application of a model-based technique to automatically generate test cases to satisfy Modified Condition/Decision Coverage (MC/DC) from high-level logic requirements expressed in a Domain Specific Language (DSL).**

I. Introduction

Many organisations in safety-critical industries including BAE Systems and General Electric Aviation Systems (GEAS) primarily use natural language (textual shall statements) to express software specification requirements. For example, the pointer shall turn red when the low height bug goes below 100ft. These textual statements are often ambiguous, untestable, incorrect, missing detail, etc. Finding this out late in the development lifecycle proves very expensive. To this end, GEAS has turned to modelling. GEAS still use textual statements to express software requirements, but supplement the requirements writing activity with modelling and simulation so that engineers gain a better understanding of the requirements and their faults. The models can then be refined and improved and serve as the Software Design artefacts. Currently this means they must write tests manually to test the design against the requirements.

Fig. 1 illustrates the current software development lifecycle. Fig. 2 illustrates the proposed changes to the software development lifecycle. The proposed methodology is to use models to express high-level requirements and then subsequently use

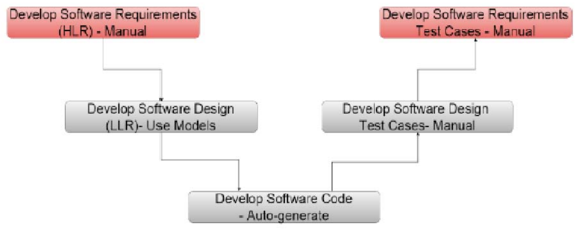

Develop Software Requirements (HLR) - Manual
Develop Software Requirements Test Cases - Manual
Develop Software Design (LLR)- Use Models
Develop Software Design Test Cases- Manual
Develop Software Code - Auto-generate

Fig. 1. Current software development lifecycle adopted by GEAS.

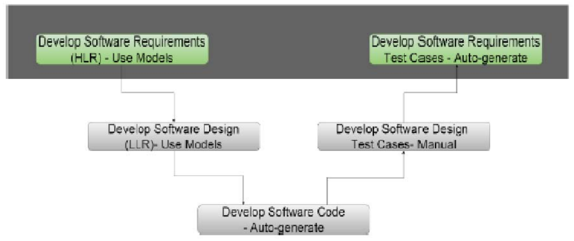

Develop Software Requirements (HLR) - Use Models
Develop Software Requirements Test Cases - Auto-generate
Develop Software Design (LLR)- Use Models
Develop Software Design Test Cases- Manual
Develop Software Code - Auto-generate

Fig. 2. Proposed changes to the software development lifecycle.

those models to auto-generate test cases that could be used to test them. This paper reports on the implementation of the proposed methodology by focusing on automated test case generation to satisfy Modified Condition/Decision Coverage (MC/DC) from high-level requirements expressed in a Domain Specific Language (DSL).

A DSL was developed using Xtext [1] to express high-level requirements. The use of a DSL for requirement modelling has several advantages as reported in [7]. The use of a textual DSL combines the benefit of natural language expressiveness while supporting model manipulation techniques. This implies that stakeholders can use a notation which incorporates domain jargon to define the functionality of software.

Software verification can consume more than 50% of the overall development cost [8]. Therefore, the automation of test generation can save time and cost of the testing process. To verify software applications in the aviation domain, certification standards such as DO-178C require the satisfaction of MC/DC coverage criteria for requirement based testing. MC/DC was developed by Chilenski and Miller to achieve a level of confidence to effectively test logical expressions without exhaustive testing [2]. The use of this criteria in functional testing has been reported to detect important errors that could not have been found at lower level tests [3]. Specification of requirements defined informally or formally can often include logic expressions to express behavioral constraints on the system. Hence, MC/DC can be employed to derive efficient tests where exhaustive testing of specifications is infeasible and also to measure the adequacy of test cases derived from logical expressions [5].

To automate the generation of requirement based tests, the proposed methodology is to generate MC/DC tests from domain specific models using model transformation techniques. Epsilon Generation Language [9] has been deployed for transforming the models to text-based test cases. This paper makes the following contributions:

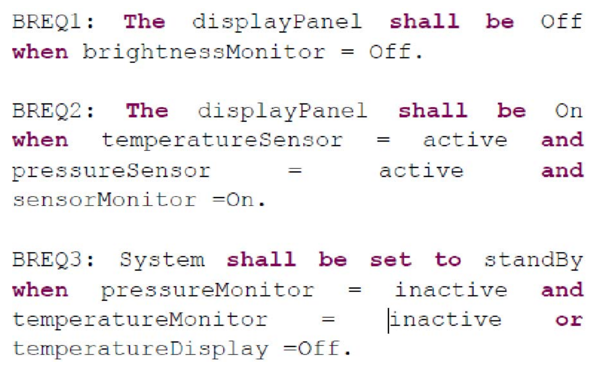
–An extension of our previous work to include automatic generation of MC/DC test cases from logic-based requirements with multiple operators using model transformation techniques.

–An evaluation of our methodology by comparing it with an existing MC/DC technique to measure adequacy of the auto-generated test cases.

The rest of the paper is organized as follows: Section II presents some related work and Section III reports on the implementation of the methodology for automated test case generation using examples from an industrial case study. Section IV evaluates the methodology by comparative analysis. Section V concludes the paper and provides some directions for future work.

II. Related Work

Several application techniques of domain-specific modelling to automated testing has been researched in the past years. These techniques have used DSLs for representing test templates and describing test scenarios. The authors of [8] developed a DSL for test case specification in the software product line domain. The language was used to describe use case models from which the test cases were automatically extracted. Puolitaival [10] also proposes a domain specific notation for test case representation. A graphical DSL is used to specify test scenarios used for automatic execution of test cases. A different approach proposed by Kanstren [11] generates a DSL from defined test models. In this case, a test model is created by the language expert using input from the domain expert. The resulting test model is then used to generate a domain specific notation for test case specification. The DSLs developed in these approaches target test case description. In this paper however, we take a different approach by the use of a DSL for requirement modelling. Also, the test cases (equivalent to test templates) are derived from the requirement models of behaviour constraints, are automatically created not manually created.


```
BREQ1: The displayPanel shall be Off
when brightnessMonitor = Off.

BREQ2: The displayPanel shall be On
when temperatureSensor = active and
pressureSensor = active and
sensorMonitor =On.

BREQ3: System shall be set to standBy
when pressureMonitor = inactive and
temperatureMonitor = inactive or
temperatureDisplay =Off.
```

Fig. 3. Logic Requirement examples.

III. Automated Test Case Generation

Logic requirements have several conditions which affect a decision. Specifications with more than one condition decision have two outcomes, one when the condition is true and another when the condition is false. The MC/DC criterion is applied when there are multiple conditions within the decision. To achieve this coverage for the requirements-based test cases, the following requirements for MC/DC are considered:

–every decision in the program has taken all possible outcomes at least once

–every condition in a decision in the program has taken all possible outcomes at least once

–every condition in a decision has shown to independently affect that decision's outcome

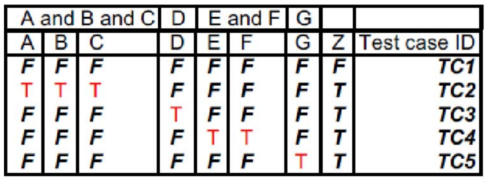
To derive test cases to satisfy the coverage criterion for specifications with multiple conditions with a single operator (AND / OR), the total number of test cases required is n+1 where n is the number of conditions in the specification [6]. Depending on the operator type, a walking false (AND) or walking true (OR) pattern is applied.

The BREQ2 requirement in Fig. 3 is used to demonstrate the walking false pattern which ensures that each condition is changed at least once for the AND operator. This pattern is implemented to give the illusion of a false value moving diagonally across the table to show that each condition independently affects the outcome of the decision. Its interpretation is done by setting each condition to false while setting fixed other possible conditions as true as shown in Table I. A walking true pattern is the opposite and gives the illusion of a true value moving diagonally while setting other conditions to false.

BREQ3 in Fig. 3 is an example of a specification with multiple operators and conditions. Behavioral requirements with multiple operators in the requirement model contain at least three conditions with combinations of AND and OR operators. To generate test cases for this type of logic requirement, the

TABLE I
WALKING FALSE TRUTH TABLE. (TS = TEMPERATURESENSOR, PS = PRESSURESENSOR, SM = SENSORMONITOR, DP = DISPLAYMONITOR)

S/N	TS=active	PS=active	SM= On	DP=On
1	T	T	T	T
2	F	T	T	F
3	T	F	T	F
4	T	T	F	F



A and B and C			D	E and F		G		
A	B	C	D	E	F	G	Z	Test case ID
F	F	F	F	F	F	F	F	TC1
T	T	T	F	F	F	F	T	TC2
F	F	F	T	F	F	F	T	TC3
F	F	F	F	T	T	F	T	TC4
F	F	F	F	F	F	T	T	TC5

Fig. 4. Overall walking true truth table.

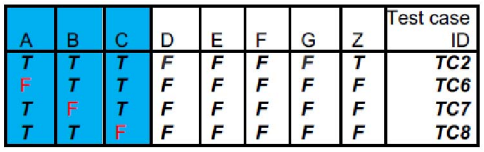
n+1 formulae is no longer adequate to ensure that MC/DC is satisfied. This is because these types of specifications introduce an additional layer of complexity with the possibility of masking conditions that could affect the outcome of the expression [4]. To achieve MC/DC satisfied test cases for multiple operator logic requirements, we employ the use of several intermediate tables.

The following generic example is used to illustrate our methodology. Z := A and B and C or D or E and F or G, where A, B, C, D, E, F. G are Boolean expressions/conditions resulting in true or false.

A. Separation of Conditions

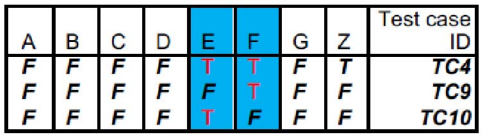
The first step is to identify the OR separated fragments in the requirement. In the context of the DSL requirement models, a fragment is a condition or combination of conditions and operators within the overall decision are treated as a unique/individual entity. This is due to the order of precedence allowing the ORs to be addressed before the ANDs. The initial requirement example can then be rewritten as: Z := (A and B and C) or D or (E and F) or G. The ANDed fragments are the conditions in brackets separated by the AND operator while non-ANDed fragments are D and G in the above example.

The first set of test cases are derived by addressing the ORed fragments in the specification and subsequent intermediate tables address the ANDed fragments of the decision. The test cases in Fig. 4 are the combination of values resulting from the application of a walking true pattern. This pattern is shown by the true value moving diagonally across the table. This is to demonstrate that each condition independently affects the outcome of the ORed expression. Its interpretation is done by varying each condition set to true while holding fixed all other possible conditions as false. In the first step, four OR-separated fragments are identified as conditions for the first set of test cases shown in Fig. 4. Using the formulae for single operator requirements, the total number of test cases is: n + 1= 4 +1 = 5. In Fig. 4, each OR-separated fragment is treated as a condition and set to true diagonally. For each test case,



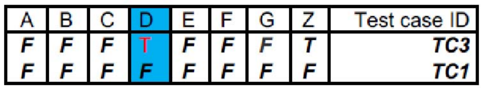
A	B	C	D	E	F	G	Z	Test case ID
T	T	T	F	F	F	F	T	TC2
F	T	T	F	F	F	F	F	TC6
T	F	T	F	F	F	F	F	TC7
T	T	F	F	F	F	F	F	TC8

Fig. 5. Walking false truth table for A, B and C.



A	B	C	D	E	F	G	Z	Test case ID
F	F	F	F	T	T	F	T	TC4
F	F	F	F	F	T	F	F	TC9
F	F	F	F	T	F	F	F	TC10

Fig. 6. Intermediate table for E and F.



A	B	C	D	E	F	G	Z	Test case ID
F	F	F	T	F	F	F	T	TC3
F	F	F	F	F	F	F	F	TC1

Fig. 7. Intermediate table for D.

the expected output from the set of values is shown in column Z.

In the first test case, all conditions are set to false and hence the expected output is false. The evaluation of other combinations of values in the table results to true. TC2 sets all conditions in the first OR-separated fragment (A and B and C) to true while holding all others to false. TC3 in the next row sets only the D condition to true while others are false. The next test case TC4 sets the conditions E and F of the next ANDed fragment to true and finally the fifth test case sets G to true.

B. ANDed fragments

This phase generates the intermediate tables for each ANDed fragment in the requirement. These fragments have more than one condition separated by the AND operator. The resulting table for each ANDed fragment is generated, containing values from applying the walking false pattern. This pattern is the opposite of walking true. For each set of conditions concerned, the pattern is shown as a false value moving diagonally across the table.

The test cases for ANDed fragments (A and B and C) is shown in Fig. 5. One can see that the first test case generated for this fragment, TC2 is a duplicate set of combinations of that of the second combination of values in Fig. 4. The test cases for the second ANDed fragment (E and F) with the application of walking false pattern is shown in Fig. 6. The first set of values in this table, TC4 is also a duplicate of the fourth set of combinations in Fig. 4.

C. non-ANDed fragments

There are normally two test cases for non-ANDed or single condition fragments. These test cases are for the situation when the conditions are true and false. The intermediate tables for the non-ANDed conditions D and G are shown in Fig.

A	B	C	D	E	F	G	Z	Test case ID
F	F	F	F	F	F	T	T	TC5
F	F	F	F	F	F	F	F	TC1

Fig. 8. Intermediate table for G.

B	C	D	Output= A	TC ID
T	T	F	T	I
F	T	F	F	II
*F	*F	T	T	III
*T	*T	T	T	IV
T	F	F	F	V

Fig. 9. Test cases from Hayhurst et.al. approach.

7 and Fig. 8 respectively. The test cases for non-ANDed conditions are normally duplicates of combinations in Fig. 4 as with the case of TC1, TC3 and TC5.

Following the generation of all the required intermediate tables, all unique combinations of values are identified. These combinations are the minimum test cases required to satisfy MC/DC for the logic requirement specification. The minimum number of MC/DC test cases for a behavior requirement specification containing multiple operators can be expressed using the formulae:

Min. no. test cases = TCI + TAC where TCI is the number of tests in the initial table (Fig. 4) and TAC is the total number of conditions in ANDed fragments. In the above example, there are five test cases in Fig. 4, three conditions in the first ANDed fragment and two in the second ANDed fragment. Therefore, there are 5 + 3 + 2 = 10 unique combinations (i.e. test cases).

IV. Evaluation

To evaluate the proposed methodology, we compared it to the Hayhurst et.al. approach [5] to determine the accuracy of the automatically generated test cases. The comparison was done by considering the example of the following Boolean expression from [5]: A : = (B and C) or D

Test cases for the above expression were automatically generated using our methodology and compared with that of the Hayhurst et.al. approach. The test cases from the Hayhurst et.al. approach are shown in Fig. 9. In TCIII, any combination of B and C resulting in a false value (i.e. FF, TF and FT) can be used to demonstrate the independent effect of D on the output. The combination of B and C to guarantee a true value (i.e. TT) is also used in TCIV. The minimum test cases required to satisfy MC/DC for the AND operator (TT, FT and FF) are TCI, TCII and TCV. For the OR operator (FF, TF and TT), they are satisfied by TCII, TCI and TCIII. Therefore, the minimum test cases to satisfy MC/DC for this example are TCI, TCII, TCIII and TCV. TCIV does not contribute to MC/DC because it is not required by any of the operators.

The test cases automatically generated using our approach are shown in Fig. 10, 11 and 12. The resulting test cases from the application of walking true on the OR-separated fragments are shown in Fig. 10. In this example, there is one ANDed fragment and one non-ANDed fragment. The test cases from applying walking false on B and C are shown in Fig. 11. The combinations for the true and false values of the non-ANDed D condition are shown in Fig. 12. After the elimination of duplicate combinations, there is a total of 5 test cases for this example.

B	C	D	Output= A	TC #
F	F	F	F	1
T	T	F	T	2
F	F	T	T	3

Fig. 10. Test cases of walking true using our approach

B	C	D	Output= A	TC #
T	T	F	T	2
F	T	F	F	4
T	F	F	F	5

Fig. 11. Test cases of walking true on ANDed fragment

B	C	D	Output= A	TC #
F	F	T	T	3
F	F	F	F	1

Fig. 12. Test cases of walking true on non-ANDed fragment

The results of our approach have produced a similar set of test cases to that of the Hayhurst et.al. approach. The minimum number of test cases required to satisfy the MC/DC criteria is 4 in both the approaches (TCI-TC2, TCII-TC4, TCIII-TC3 and TCV - TC5). The outstanding test cases in both Hayhurst et.al. and our approaches are TCIV and TC1 respectively. These test cases could be included but are not necessary to satisfy MC/DC.

A. Illustrated Example

We apply our methodology to 2 different requirement sets from our industry partner. The first set of requirements described in the language was for a Power Unit (SubsystemA) in the system which consists of 32 input signals, 3 definition requirements and 25 logic-based behaviour requirements. The language was also used to represent a second set of requirements for Annunciations (SubsystemB) in the system which consists of 15 input signals, 81 definition requirements and 14 behaviour requirements. The behaviour section of SubsystemB is made up of 8 logic-based requirements and 6 conditional based statements of which the test case generation is beyond the scope of this paper. Table II shows the results of automatic test case generation from the specifications of SubsystemA and SubsystemB. In the case of SubsystemA, a total of 139 MC/DC test cases were generated from 25 requirements in 10078ms with an average of 72.5ms per test case.

TABLE II
MC/DC TEST CASE GENERATION TIMES

	Total logic requirements	**Cumulative total test cases**	**Total time taken**
SubsystemA	8	40	2907ms
SubsystemB	25	139	10078ms

V. Conclusions and Future Work

This paper has discussed a methodology to automate the test case generation of high-level requirement specifications. High-level requirements are expressed in a DSL. Model transformation techniques are then applied to transform the requirements into textual test cases. The paper has focused on the generation of MC/DC test cases from logic requirements consisting of multiple logical operators. The accuracy of the methodology has been evaluated by comparing it with an existing manual approach. Future work would involve the application of model transformation techniques to other types of requirement specifications including pseudo requirements, boundary value analyses and timing requirements.

Acknowledgment

This work is part of a project that is jointly funded and supported by both University of Northampton and GE Aviation, Cheltenham.

References

[1] L. Bettini, *Implementing domain-specific languages with Xtext and Xtend* Packt Publishing Ltd, 2016.

[2] J. J. Chilenski and S. P. Miller, *Applicability of modified condition/decision coverage to software testing* Software Engineering Journal, 9(5), 193-200, 1994.

[3] A. Dupuy and N. Leveson, *An empirical evaluation of the MC/DC coverage criterion on the HETE-2 satellite software*, In Digital Avionics Systems Conference, 2000. Proceedings. DASC. The 19th (Vol. 1, pp. 1B6-1), IEEE, 2000.

[4] K. J. Hayhurst, D. S. Veerhusen, J. J. Chilenski and L. K. Rierson, *A practical tutorial on modified condition/decision coverage*, National Aeronautics and Space Administration, Langley Research Center, 2001.

[5] K. J. Hayhurst and D. S. Veerhusen, *A practical approach to modified condition/decision coverage*, In Digital Avionics Systems, 2001. DASC. 20th Conference (Vol. 1, pp. 1B2-1), IEEE, 2001.

[6] P. Mitra, S. Chatterjee and N. Ali, *Graphical analysis of MC/DC using automated software testing* In Electronics Computer Technology (ICECT), 2011 3rd International Conference on (Vol. 3, pp. 145-149). IEEE, 2011.

[7] O. Olajubu, *A textual domain specific language for requirement modelling*. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (pp. 1060-1062), ACM, 2015.

[8] S. Rayadurgam and M. P. Heimdahl, *Generating MC/DC Adequate Test Sequences Through Model Checking*. In SEW (p. 91), 2003.

[9] L. M. Rose, R. F. Paige, D. S. Kolovos and F. Polack, *The epsilon generation language* ECMDA-FA, 8, 1-16, 2008.

[10] O. P. Puolitaival, T. Kanstren, V. M. Rytky, and A. Saarela, *Utilizing domain-specific modelling for software testing*, In 3rd International Conference on Advances in System Testing and Validation Lifecycle, pages 115-120, 2011.

[11] T. Kanstren and O. P. Puolitaival, *Using built-in domain-specific modeling support to guide model-based test generation*, Electronic Proceedings in Theoretical Computer Science, 80:58-72, 2012.