A New Comment on Reinforcement of Testing Criteria

Monika Singh
College of Engineering and Technology,
Mody University of Science and Technology,
Lakshmangarh, Rajasthan, India

Vinod Kumar Jain
College of Engineering and Technology,
Mody University of Science and Technology,
Lakshmangarh, Rajasthan, India

***Abstract*—This paper presents the formal aspects of testing criteria for Safety Critical Systems. A brief review of testing strategies i.e. white box and black box is given along with their various criteria's. Z Notation; a formal specification language is used to sever the purpose of formalization. Initially, the schemas are formed for Statement Coverage (SC), Decision coverage (DC), Path Coverage (PC), Equivalence Partition Class (EPC), Boundary Value Analysis (BV) and Cause & Effect (C&F). The completeness and correctness of test schema are enriched by verifying these with Z/EVES; a Theorem Prover tool for Z specification.**

***Keywords*—Formal Methods; Safety Critical System; Z Notation; Schema**

I. Introduction

Testing [1] plays an important role for checking the correctness of system implementations. To test system, test cases are formed and system behavior has been observed during execution. Based on test execution, the decision is made for the correctly functioning of the system. However, the criterion for the correctness of test cases has been specified in the system specification. A specification prescribes "What" part of the system i.e. the function that a system supposes to do and accordingly forms the foundation for testing criteria. As system specifications are documented in natural language (informal), which is generally incomplete and ambiguous in nature, due to this many problems may occur in testing processes such as incompleteness, ambiguous and inconsistency in test specifications. With an unclear specification, it is next to impossible to predict how the implemented system will behave; consequently testing will be difficult as it is not clear what to test. This become more severs specifically in case of Safety Critical System [2]. An ambiguous system specification which further forms the root for test specification may raise many problems such as misinterpretation and therefore needs explanations of specification's purpose. This requires rework of the system specification during the testing phase of software development. The rework process takes too much time, money and efforts which ultimately delay the process of deployment of system. Therefore, there is an utter need of usage of the formal model [3] for testing criteria of Safety Critical Systems [4] for test case's completeness and correctness. Formal methods are equipped with rich mathematical axioms and tool support. This rich tool support will help further verification of test specification in automated environment. In this paper, the purpose of formalization has been accomplished by Z Notation [5] and simulation has been done with Z/EVES [6]: an automated Theorem Prover.

Formal methods: Formal methods [3] are the methods which use mathematical techniques as their foundation pillars and are used to develop the software systems. They can be applied at any phase of software development process, but highly recommended to apply in early phases. By using formal methods, one can reduce the chances of ambiguities and incompleteness in requirements documents, design specification and the test case specification. There is a range of formal specification languages available to design the software system such as Z notation [5], B-methods [7], VDM [8] etc which are further verified by Theorem Prover [9] and Model Checker [9]. Broadly, formal methods are categorized into two groups:

a) Model based Formal methods: In this group, the formal specifications are consisting of mathematical structures such as relations, functions, sets and sequences to design software system model. The members of this group are: Z Notation [5], VDM [8], B-Methods [7], Petri net [10], Communicating Sequential processes (CSP) [11].

b) Property oriented formal methods: Property oriented formal methods, on the other hand, the specifications of system are defined in terms of its properties, generally in form of axioms which satisfied by the system. For example, OBJ, LOTOS [12], Larch lies in this group.

In this paper, we use Z notation to write done the test documents which is further analyzed by Z/EVES Theorem Prover tool. Rest of the paper is organized as follow: Section 2 represents the methodology and research components. Section 3 presents the Formal aspect of testing strategies for Safety Critical Systems. Section 4 advocates the simulation results and discussions. At last, the conclusion is given base on section 4 analysis in Section 5.

II. Methodology and Research Components

Initially the schemas of testing criteria i.e. SC, DC, PC, EPC, BV and C&F are formed by using Z Notation. Once the schemas are formed, they are checked for their completeness and correctness using Z/EVES; automated Theorem Prover tool for Z specification. If errors occur, corrections are made in respective schema and again execute on Z/EVES. This process is repeated until error free schemas are come as an output. Figure 1 presents the formal model of testing strategies which composed of following research components:

A. *White Box testing*

White box testing [13] is more concerned about the implementation details such as: programming style, control methods: statements coverage, decision coverage, condition coverage etc. It is also known as structural testing. It emphasizes on internal structure of software artifact. The internal structure mainly tested by using the following scenarios:

- Statement coverage: Test cases are executed in such a way that all statements have been covered once.
- Branch/decision coverage: Test cases are executed in such a way that both if-branch and else –branch covered.
- Path coverage: Test cases are executed in such a way that each possible path has been executed once.

B. *Black Box Testing*

Black box testing [14] focuses on functional/ behavioral testing of system without peeking into internal structure of system. It is also known as functional testing. It can be done by following ways:

- Equivalence partition classes: The input set is partitioned into equivalence classes and a single test case is executed for each class. The single test case is valid for all the elements of a given class. However, the classes chosen should be disjoint to avoid redundancy.
- Boundary value Analysis: In boundary value analysis rather than taking input from the partition classes, test cases are executed for boundary value points or near the boundary of partition classes.
- Cause & Effect Graph: In cause and effect (CF) graph, the combinations of inputs are analyzed. The cause is a representation of inputs and effect is a symbol of resultant output. Boolean graphs are used to link various causes and their respective effects.

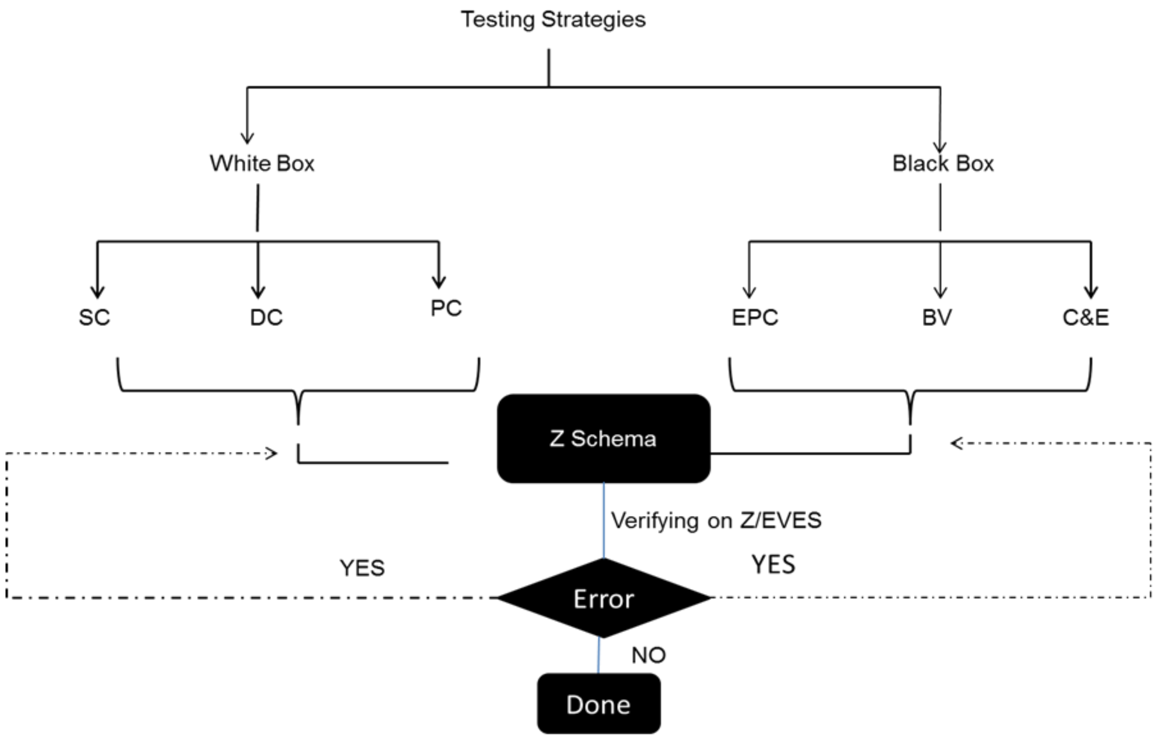


Fig. 1. Formal model of Testing Strategies

C. *Z Schema*

Schema is the notion used to structure the specification written in Z notation. It's composed of three parts: schema name, variable and constraints.

The generic structure of schema which showed in figure 2 consists of three parts as:

- Schema Name
- Variables declaration
- Constraints

_*SchemaName*

Variables declaration

constraints (*preconditions or postconditions*)

Fig. 2. Basic Schema structure

D. *Z/EVES*

Z/EVES toolset is an interactive tool for composing, checking, and analyzing Z specifications. It is based on the

EVES system, and uses its proof checker to carry out its proof steps. The language accepted by Z/EVES is a LATEX markup form [19]. This toolset helps in the analysis of Z specifications in several ways: (1) syntax and type checking, (2) schema expansion, (3) precondition calculation, (4) domain checking, (5) and general theorem proving [7]. The model checker of the Z/EVES is considered as user friendly and simple, especially when compared with other related tools such as Isabelle-HOL or Proof Power-Z. It could also prove its merit and popularity; due to its power in proving the specifications of critical systems written using the Z notation.

III. Formal Aspect of Testing Strategies

This section composed of two parts: Formal transformation of White box testing and Formal transformation of Black box testing.

A. Formal transformation of White box testing

White box testing focuses on internal structure of software artifact. One of the ways to test internal structure is to use either of following scenarios: Statement coverage, decision coverage or path coverage (Figure 1). However, various definitions of these scenarios may raise ambiguity. One possible solution is the elaboration of formalized definition of testing criteria by using rigorous mathematics such as set theory graph theory, predicates logics etc. In this paper, Z notation (Formal Specification Language) has been used to serve the purpose.

To check the completeness and correctness of above mention scenarios, mathematical structure i.e. Z –schema has been used. For any testing criteria, the two basic sets are required i.e.

[INPUT, STATEMENT]

Where INPUT is the set of all possible values of input variable and STATEMENT is the set of all program statements. Since in Statement coverage, every statement in the program has been executed at least once, therefore we define a function *path* from INPUT to STATEMENT as

Path i: INPUT →STATEMENT

Along with path function, we need to define two other set i.e. BOOL, INPUT-PART and COND as follow:

BOOL= {0, 1}

Which are respective values of executed conditions (as 1) and non-executed conditions (as 0)

INPUT-PART= P INPUT \ {INPUT}

i.e. non-empty set of input variables which yet not executed. Moreover, INPUT-PART is a subset of INPUT.

COND is a non-empty set which contain values by mapping an input i ∈ INPUT to true condition (as 1) and false condition (as 0).

COND== INPUT → BOOL

Now the formal definition of Statement coverage is given by using Z schema as:

```
________________SC
| decinput !: $\mathbb{P}_1$INPUT
| decst ?: STATEMENT
| decinput 0, decinput 1: INPUT-PART
|_
| decinput = {i: INPUT |decst ∈ path i}
| <decinput 0, decinput 1> partitions decinput
| Dom value = decinput
|
```

The constraints are: (i) the domain all values should be the set of input; (ii) The input values partition the set of input and (iii) For each i, the function path maps the input to respective statement. Therefore based on this, test_ data has been built which satisfy or not satisfy testing criteria.

Now the Decision Coverage (DC) is formally defined as:

```
______________________DC
| ΔSC
|_
| ∀ d:dec • (test-data ∩ d •  decinput 0 ≠ ∅) ∧
| (test-data ∩ d• decinput 1) ≠ ∅
|________________________________
```

The constraints of DC schema are defined as: if there is if-else condition, both of the decision will execute which consequently satisfy the definition of decision coverage. However, there would be change in statement coverage schema which has been shown by ΔSC.

The next schema is Path Coverage (PC)

```
_PC
| ΔSC
| ΔDC
|_
| ∀ i, j ∈ℕ, (path i ∩ path j = ∅) ∧
| (path i ∪ path j = test-data)
|________________________________
```

The constraints of DC schema are: (i) all the possible paths are covered at least once and if and two paths are identical.

B. Formal Transformation of Black box testing

The three black box testing criteria which are considered here are:

- Equivalence partitions class (EPC)
- Boundary Value Analysis (BV)
- Cause & Effect (C&E)

As mentioned in section 2 (b), the test data is partitioned into equal classes and for each class only one value from test data is tested. Therefore for schema, two basic sets are:

[TEST_DATA, CLASS]

Now the schema of EPC is as follow:

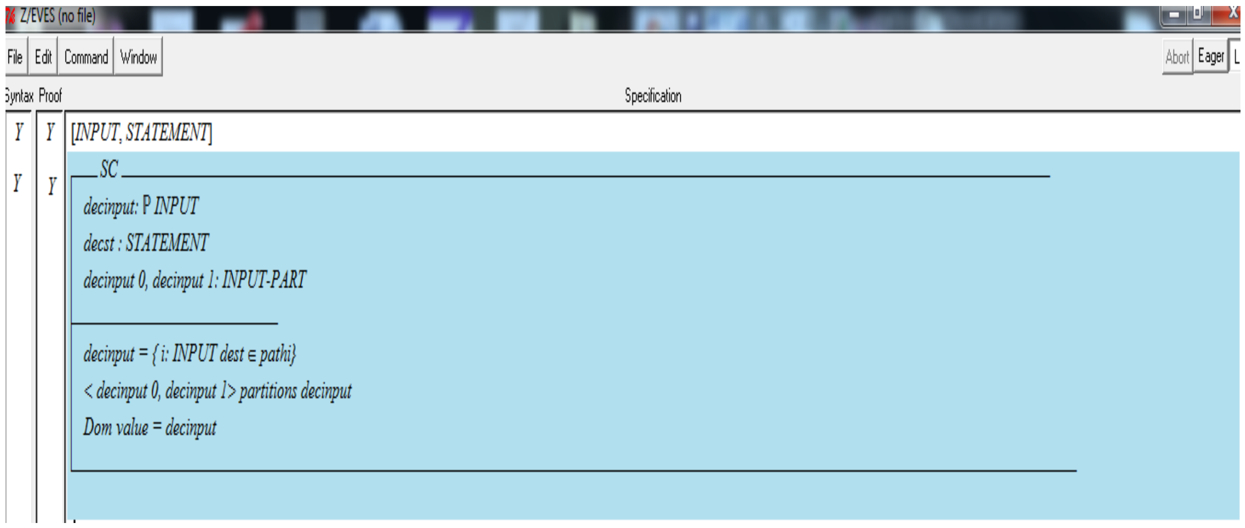
```
_________EPC
| tstdat: TEST_DATA
| cls: CLASS
|__
| ∀ i, j∈ ℕ, tst1, tst 2 ∈ TEST_DATA ∧
| tst 1 ∩ tst 2 = ∅ ∧
| ∀ i ∈ ℕ, cls1, cls2 ∈ CLASS ∧
| cls1 ∩cls2 = ∅ ∧
| ∪cls i = CLASS
|________________________________________
```

The constraints are: All the partitions are disjoint and one test value should be chosen from one class. For Boundary value analysis, the boundary values of each class are tested. In other words, we need to test the five values for each class i.e. (i) Minimum (ii) Just above the minimum (iii) A nominal value (iv)Just below the maximum and (v) Maximum. Therefore the schema for boundary value is:

```
_________BV
| Δ EPC
|__
| min, max, nominal ∈ TEST-DATA ∧
| jst-abv-min, jst-blw-max ∈ TEST-DATA
|________________________________________
```

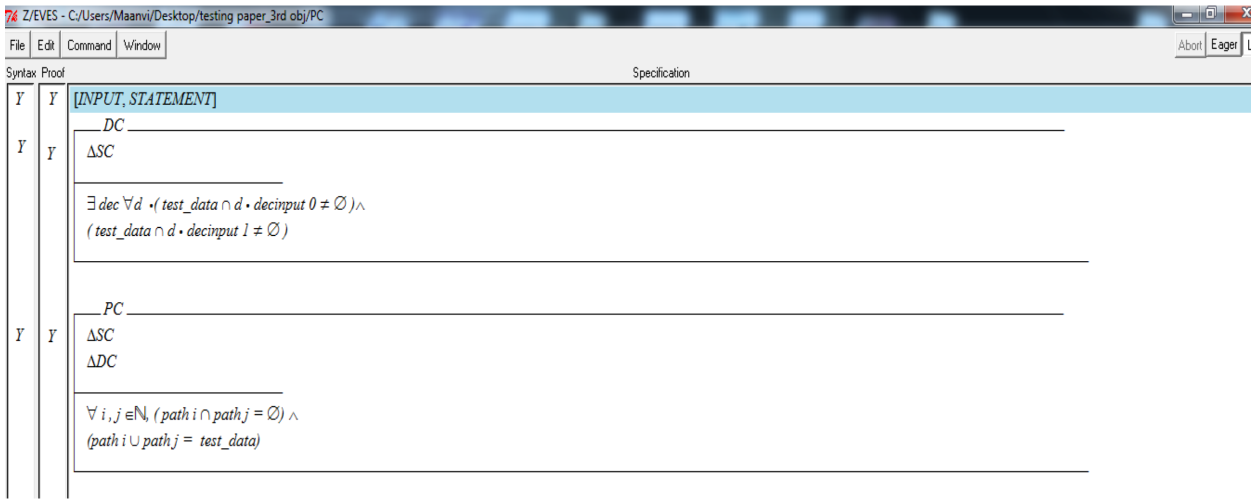
IV. Simulation and Discussion

Although the formal specification languages uses mathematics notation (in this paper Z notation has been used), yet chances of ambiguities are still there. Automated or semi-automated tool are used to check the Z specification. Z/EVES; a Theorem Prover tool for syntax, type checking and domain checking is used for checking the Z specification. The graphical interface of Z/EVES tool consists of two columns: Syntax and Proof. The columns with 'Y' value show that there is no error. Once the specification written, the file has been stored with extension ".zev". Figure 3 depicts the execution of Statement Coverage (SC) specification for syntax and type checking. It is cleared from figure 3 that both the columns have value 'Y' indicating that SC schema is free from syntax and domain errors. Similarly, Fig. 4, 5 represents the formal part of Path coverage (PC) and Equivalence Partition Class (EPC) respectively.



```
[INPUT, STATEMENT]
___SC_____________________________
| decinput: ℙ INPUT
| decst : STATEMENT
| decinput 0, decinput 1: INPUT-PART
|__________________
| decinput = { i: INPUT dest ∈ pathi}
| < decinput 0, decinput 1> partitions decinput
| Dom value = decinput
|__________________________________
```

Fig. 3. Formalization of SC schema

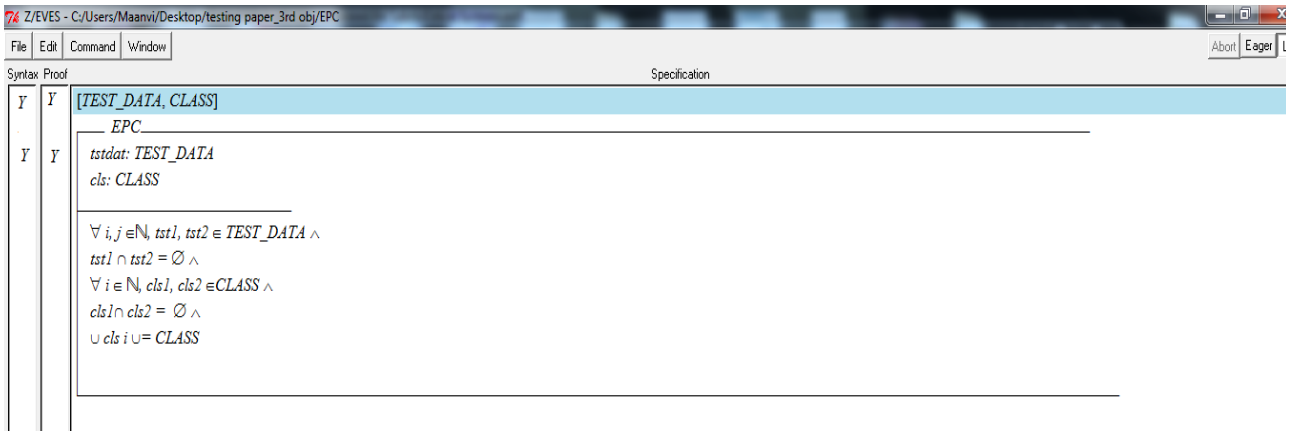


```
[INPUT, STATEMENT]

__DC________________________________
ΔSC
____________________
∃ dec ∀d •( test_data ∩ d • decinput 0 ≠ ∅ )∧
( test_data ∩ d • decinput 1 ≠ ∅ )

__PC________________________________
ΔSC
ΔDC
____________________
∀ i , j ∈ℕ, ( path i ∩ path j = ∅) ∧
(path i ∪ path j =  test_data)
```

Fig. 4. Formalization of Path Coverage specification



```
[TEST_DATA, CLASS]

__EPC________________________________
tstdat: TEST_DATA
cls: CLASS
____________________
∀ i, j ∈ℕ, tst1, tst2 ∈ TEST_DATA ∧
tst1 ∩ tst2 = ∅ ∧
∀ i ∈ ℕ, cls1, cls2 ∈CLASS ∧
cls1∩ cls2 =  ∅ ∧
∪ cls i ∪= CLASS
```

Fig. 5. Execution of EPC for syntax and Domain checking

V. Conclusion

The main idea of this article is formalization of testing criteria for safety critical systems. Software Testing Techniques are broadly partitioned into two groups i.e. white box testing and black box testing. For white box testing, three criteria's are used i.e. Statement Coverage (SC), condition Coverage (CC), Path Coverage(PC) and for Black Box testing, the criteria's which has been used are Boundary Value Analysis (BV), Equivalence Partition (EP)class, Cause & Effect (C&E). All these criteria used to figuring out the branch and loop structure using logical expressions in program. For fulfilling the definition of formalization, Z notation is used. Initially Z schemas are formed for each criterion's .i.e. SC, PC, BC, EP, BV and C&F. To check the correctness and completeness of schemas, Z/EVES tool is used further. The findings of Z/EVES are syntax checking, domain checking and type checking.

References

[1] Glenford J. Myers. The Art of Software Testing, John Wiley & Sons, 2nd Edition, 2004.

[2] IPL. An Introduction to Safety Critical System, executive summary, 1997.

[3] Monin, Jean-Francois. Understanding Formal Methods, 2003, Springer.

[4] Jonathan Bowen. Safety Critical Systems, Formal Methods and Standards, 1992, Software Engineering Journal.

[5] J. Michael Spivey. The Z Notation: A Reference Manual, 2001, 2nd eds. Prentice Hall.

[6] Saaltink, M. The Z/EVES 2.0 User's Guide, Technical Report TR-99-5493-06a, ORA Canada, One Nicholas Street, Suite 1208 - Ottawa, Ontario K1N 7B7 - CANADA, 1999.

[7] S. Schneider. B Method- an Introduction Palgrave, Cornerstones of Computing series, 2001.

[8] C. B. Jones. Systematic Software Development using VDM, 1990, In Prentice Hall.

[9] Hasan Amjad. Combining model checking and theorem proving, 2004, technical report, University of Cambridge, computer Laboratory. UCAM-CL-TR-601, ISSN 1476-2986.

[10] Reisig, Wolfgang. Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies, 2013, 1st Eds. Springer-Verlag Berlin Heidelberg, ISBN 978-642-33277-7.

[11] C. A. R. Hoare. Communicating Sequential Processes, 1985, In Prentice Hall.

[12] Howard Bowman. A LOTOS based tutorial on formal methods for object-oriented distributed systems, New Generation Computing, Vol. 16, Issue 4, pp 343-372.

[13] Glenford J. Myers, Corey Sandler, Tom Badgett. The Art of Software Testing, 2011 3rd Eds. Wiley.

[14] Paul C. Jorgensen. Software Testing: A Craftsman's Approach, 2013, 4th Eds. Auerbach Publications.