CHAPTER 2

STATE OF THE ART IN SPECIFICATION-BASED TESTING

A comprehensive state of art has been presented and the salient features have been imbibed to advance the knowledge in this area.

2.1 TEST DATA GENERATION FROM BOOLEAN SPECIFICATION

Weyuker et al (1994) have presented a formal description of the meaningful impact strategy for testing the implementation of a boolean formula and have analyzed the fault detection ability for both Disjunctive Normal Form (DNF) and Canonical Disjunctive Normal Form representation of a boolean formula. They examined the effectiveness of their strategies empirically using a set of twenty specifications taken from the specification for a real Traffic Alert and Collision Avoidance Subsystem (TCAS II). The empirical results on the fault detecting ability of their family of strategies are very encouraging.

2.1.1 Basic Meaningful Impact Strategy

The intuition underlying this approach is that, given a representation of a boolean formula, a test set should be chosen such that each literal occurrence in the representation demonstrates its meaningful impact on the outcome. This strategy is testing directly for one particular type of fault, namely, Variable Negation Fault ($VNF$). The faulty implementation of the boolean formulae given in the boolean specification are detected by the strategies proposed by them.
Their study was fault based and they used substantially larger-size formulae and designed algorithms to automatically generate test sets to test any proposed implementation of a given boolean specification. They have also defined six variants of the basic meaningful impact strategy and evaluated their effectiveness at detecting five distinct type of faults.

When testing boolean formulae in irreducible DNF, the following test cases are required by the basic meaningful impact strategy.

i. Select one test point from each non-empty Unique True Point (UTP$_i$) of each term in the formula. A point selected from UTP$_i$ demonstrates the meaningful impact of each of the literals in product term P$_i$ on the 1 outcome, because all the product terms other than P$_i$ will evaluate to 0.

ii. Select one test point from Near False Point (N$_i,j$) of F. A point picked from N$_i,j$ demonstrates the meaningful impact of the literal I$_j$ on the 0 outcome, because, for such a point all other literals in the product term P$_i$ evaluates to 1.

The definition of the sets U$_i$ and N$_i,j$ define test conditions that must be satisfied. Their tool form the sets U$_i$ and N$_i,j$ and when necessary, O (Overlapping True Point) and M (Remaining False Point), it selects points from those formally defined sets.

### 2.1.2 Enhancing the Basic Strategy

They have introduced a family of algorithms for automatically generating test cases for implementations intended to satisfy specifications that are boolean formulae. Four of the strategies enhance the basic impact strategy by increasing the number of test cases selected from each set, and two of those strategies are sample sets
not sampled in the basic meaningful impact strategy. The number of test cases required by the strategy increases.

While selecting points from a set, the selection is done randomly by using a uniform distribution. Once a point is selected, it is removed from consideration from all other sets. The first two variants essentially implement the basic meaningful impact strategy. ONE is a straightforward implementation of the strategy and MIN attempts to optimize the ONE strategy.

2.1.3 MIN

One point is selected from the set of Unique True Points associated with each term, and the minimum set of points needed to satisfy the basic meaningful impact strategy are selected from the set of Near False Points of the formula.

2.1.4 ONE

One point is selected from the set of Unique True Points associated with each term, and one point is selected from each set of Near False Points, $N_{ij}$. In the following variants of the meaningful impact strategy, more than one point is selected from a given set. In most cases, the number of points selected is determined by the size of the set being sampled.

2.1.5 MANY-A

For a set of $2^X$ Unique True Points associated with a term, $[X]$ points are selected from the set, and $[X]$ points are selected from each set of Near False Points, $N_{ij}$ of size $2^X$. If $X=0$ for some set, then one point is selected from that set.
2.1.6 MANY-B

For a set of \(2^X\) Unique True Points associated with a term, \([X]\) points are selected from the set, and \([X]\) points are selected from each set of Near False Points, \(N_{i,j}\) of size \(2^X\). In addition, \([X]\) points are selected from the set of Overlapping True Points of size \(2^X\) and \([X]\) points are selected from the set of Remaining False Points of size \(2^X\). If \(X=0\) for some set, then one point is selected from that set.

2.1.7 MAX-A

Every point of the set of Unique True Points associated with each term is selected, and every point from each set of Near False Points, \(N_{i,j}\) is selected.

2.1.8 MAX-B

Every point of the set of Unique True Points associated with each term is selected, and every point from each set of Near False Points, \(N_{i,j}\) is selected. In addition, \([X]\) points are selected from the set of Overlapping True Points of size \(2^X\), and \([X]\) points are selected from the set of Remaining False Points of size \(2^X\). If \(X=0\) for some set, then one point is selected from that set.

2.2 TEST DATA SELECTION STRATEGIES FOR BOOLEAN SPECIFICATION

Chen et al (1997) proposed two test data selection strategies towards testing a boolean specification. They mentioned that MAX- B strategy proposed by Weyuker et al (1994) guaranteed the detection of faults. However the strategy proposed by Chen et al (1997) was more effective than MAX-B strategy with respect to the detection of Literal Insertion Fault and Literal Reference Fault, in the sense that test cases selected by their strategy were a subset of those selected by MAX-B.
strategy. They mentioned that the Basic Meaningful Impact Strategy does not guarantee to detect Literal Insertion Fault and Literal Reference Fault. They discussed test data generation using a fault-based approach in the sense that generation of test cases is based on particular type of faults occurring in the boolean expression. They proposed two independent and complementary test case selection strategies for testing boolean specifications. They defined a number of faults that could appear in a boolean formulae, such as Literal Insertion Fault, Literal Reference Fault, Literal Omission Fault, Term Omission Fault, Literal Negation Fault and Expression Negation Fault. They mainly concentrated on detecting Literal Insertion Faults and Literal Reference Faults. They said that the test case to detect Literal Insertion Fault always occurred in the set of Unique True Points. They have also mentioned that Literal Reference Faults could be hypothesized from Near False Points and Unique True Points. They have also pointed out that the Basic Meaningful Impact Strategy cannot guarantee to detect the Literal Insertion Fault \( (LIF) \) and Literal Reference Fault \( (LRF) \).

The Literal Insertion Fault can be detected by Unique True Points because both \( S \) and \( I_{ij} \) agree on all Near False Points, all Overlapping True Points and all Remaining False Points. \( LRF \) can only be detected by Unique True Points and Near False Points because any Overlapping True Point cannot detect \( LRF \) and any Remaining False Points cannot detect \( LRF \).

2.2.1 MUTP Strategy

The Multiple Unique True Point (MUTP) Strategy, selects test cases from a set of Unique True Points. This strategy guarantees the detection of \( LIF \). If the literal \( x_L \) evaluates to the same truth value \( 1(0) \) for all Unique True Points in \( UTP_j(S) \) where \( x_L \) does not occur in \( P_i \), we do not need to consider the Literal Insertion Fault for the literal \( x_L \) that will be made on \( P_i \) because \( S \) and \( I_{ij} \) are equivalent. Two elements in \( UTP_j(S) \) are required such that the literal \( x_L \) evaluates to 1 and 0. The literal \( x_L \) is
defined as a missing – disagreed literal with respect to the term $P_i$ of the boolean expression $S$ if

i. $x_L$ does not occur in $P_i$ and

ii. $x_L$ does not evaluate to the same truth value for elements in $UTP_i(S)$

The MUTP strategy selects Unique True Points from every $UTP_i(S)$ such that every missing – disagreed literal with respect to $P_i$ takes on 1 and 0. Clearly the MUTP strategy guarantees to detect $LIF$.

2.2.2 CUTPNFP Strategy

The Corresponding Unique True Point and Near False Point (CUTPNFP) strategy guarantees the detection of $LRF$. The CUTPNFP strategy requires a pair of Unique True Point and Near False Point be selected from each $UTP_i(S)$ and $NFP_{i,j}(S)$ pair respectively. The pair is chosen such that they differ only in the corresponding value of the $j^{th}$ literal $x_{i,j}$ of the $i^{th}$ term $P_i$ in $S$.

If the corresponding pair of Unique True Point and Near False Point does not exist, a simple checking to see whether all Unique True Points agree on a particular literal $x_t$ can determine whether $S$ and $I$ are equivalent.

If $S$ and $I_{id}$ are not equivalent, then either (i) a corresponding pair of Unique True Point and Near False Point, that satisfies the condition exists or (ii) the literal $x_t$ not appearing in $P_i$ does not evaluate to 1 for all Unique True Points in $UTP_i(S)$. If the Corresponding Unique True Point and Near False Point pair exists then $LRF$ can be detected, otherwise, use the MUTP strategy that guarantees to detect the $LRF$. Whenever the CUTPNFP strategy cannot find a corresponding pair of Unique True Point and Near False Point, apply MUTP strategy in order to detect the corresponding $LRF$. The CUTPNFP strategy does not guarantee to detect $LIF$. 
2.3 A NEW TEST DATA SELECTION FOR TESTING BOOLEAN SPECIFICATION

Noritaka Kobayashi et al (2001) proposed a new test data selection strategy for testing boolean specifications. They observed that exhaustive testing is infeasible because the number of possible test cases is prohibitively large. In their note, they discuss the testing of an implementation, intended to satisfy a given specification, that is a boolean formula. Many portions of the behaviour of a software system can be represented as a boolean formula. They have considered two strategies given by Weyuker et al (1994), (i) the Basic Meaningful Impact Strategy (BMIS) (ii) the MAX-A strategy, which is the most powerful strategy, since the strategy guarantees the detection of many types of faults.

They propose a new test data selection strategy, which also guarantees the detection of all the types of faults. They have showed that the proposed strategy can achieve the same effectiveness as MAX-A with much lower cost.

They have considered black – box testing of an implementation of a specification, that is a boolean formula. The boolean formulae $S$ and $F$ denote a specification and its faulty implementation. Variables in the formula represent input parameters that take boolean values. A fault will be detected iff, with a test case, $F$ evaluates to a different value than $S$.

Basic Meaningful Impact strategy (BMIS) proposed by Weyuker et al (1994) selects test cases such that one point is selected from every non-empty $\text{UTP}_i(S)$ and $\text{NFP}_{ij}(S)$. On the other hand, MAX-A selects all points of every non-empty $\text{UTP}_i(S)$ and $\text{NFP}_{ij}(S)$.

The proposed strategy namely Extended Meaningful Impact Strategy (EMIS) guarantees the detection of seven types of faults such as $TNF$, $LNF$, $ORF$, ...
TOF, LOF, LIF, LRF. Although MAX-A also guarantees the detection of all these types of faults, EMIS achieves more efficiency than MAX-A by selecting test cases from each $\text{UTP}_i(S)$ and $\text{NFP}_i(S)$ appropriately. EMIS selects points based on the following two steps.

i. Points are selected from each $\text{UTP}_i(S)$ such that, as many values for variables in $V-V_1$ as possible are covered while minimizing the number of selected points. The set of variables of $S$ and the set of variables of $P_i$ are denoted by $V$ and $V_1$ respectively.

ii. Points are selected from each $\text{NFP}_i(S)$ when there are variables in $V-V_1$ that do not take 0(1) in the points selected from $\text{UTP}_i(S)$ in the first step, new points are selected from $\text{NFP}_i(S)$ such that as many of these variables as possible take 1(0) while minimizing the number of selected points. If no point can be selected this way, one arbitrary point is selected.

They have shown the effectiveness of their method by applying it to Literal Reference Fault and explained that their method ensures fault detection. They have used mutation analysis to demonstrate their claim. By adopting the seven fault models, they have created 51 mutants for the specification $S = abd + cd + e$. They have compared their method with BMIS and MAX-A strategies. The mutation score obtained by BMIS, MAX-A and EMIS were 86.3%, 100% and 100% respectively, while the sizes of test suites generated by BMIS, MAX-A and EMIS were 10, 26 and 11. Although both EMIS and MAX-A guarantee the detection of all types of faults, the result shows that EMIS is more effective than MAX-A.
2.4 FAULT DETECTION IN SPECIFICATION-BASED TESTING

Kuhn (1999) presented a method for computing the conditions that must be covered by a test set for the set to guarantee detection of the particular fault class. He has also shown that there is a coverage hierarchy of fault classes that is consistent with the experimental results on fault based testing.

Various faults are defined in the literature.

1. Variable Reference Fault
2. Variable Negation Fault
3. Expression Negation Fault
4. Associative Shift Fault
5. Operator Reference Fault
6. Incorrect Relational Operators
7. Incorrect Parenthesis
8. Incorrect Arithmetic Expressions
9. Extra Binary Operators
10. Missing Binary Operator

The detection conditions for a predicate $P$ are the conditions under which a change to $P$ will affect the value of the predicate $P$. A test will detect a failure if and only if a faulty predicate $P'$ evaluates to a different value than the correct predicate $P$, that is, $P \oplus P'$, where $\oplus$ is exclusive – or. Faults can be analyzed by letting $P'$ be the predicate $P$ with the faults inserted. Given a particular fault, hypothesized for a particular specification, it is possible to compute the conditions under which the fault will cause a failure. That is, conditions under which the fault will cause the expression to evaluate to a different value than if the fault had not occurred. To determine, for example, the condition under which a Variable Negation Fault for variable $v$ will be detected, we simply compute $P \oplus P_v^*$, where $P_v^*$ is the predicate $P$ with all free
occurrences of variable $x$ replaced by expression $e$. The hierarchy of fault classes is based on the conditions under which a particular type of fault is detected. It is shown that this hierarchy can be used to explain the empirical results of fault-based testing.

2.4.1 Fault Classes

The detection conditions for the various fault classes are determined as follows. Let $S$ be a specification in Disjunctive Normal Form.

$$S = x_1 \land x_1 \land \ldots \lor x_2 \land x_2 \land \ldots \lor x_n \land x_n \land \ldots$$

In general, the $x_i$ variables may not be distinct. The conditions under which a Variable Negation Fault for variable $a$ will be detected are $S \oplus S^a_S$. The condition for detecting Variable Negation Faults ($SVNF$), Variable Reference Faults ($SVRF$) and Expression Negation Faults ($SENF$) are given.

$$SVNF = S \oplus S^{a_{i_j}}$$
$$SVRF = S \oplus S^{x_{k_i}}$$, where $x_{k_i}$ is the variable substituted for $x_{i_j}, x_{k_i} \neq x_{i_j}$
$$SENF = S \oplus S^{X_i},$$ where $X_i$ is the clause $x_i \land x_i \land \ldots \land x_i$.

It can readily be shown that $SVRF \Rightarrow SVNF \Rightarrow SENF$ under very minimal restrictions. Kuhn (1999) has presented a theorem showing the relationship between the test for Variable Reference Fault and Variable Negation Fault. If the variable replaced in $SVRF$ is the same variable negated in $SVNF$, then $SVRF \Rightarrow SVNF$. Kuhn (1999) has given two cases for showing the above relationship.

Case 1: Substituting a variable from a different clause.
Case 2: Substituting a variable from the same clause.
He has also shown the relationship between the test sets for Variable Negation Fault and Expression Negation Fault. If expressions containing the variable negated in $SVNF$ are negated in $SENF$ then $SVNF \Rightarrow SENF$.

In terms of test conditions, the relationship between the fault classes is $SVRF \subseteq SVNF \subseteq SENF$. Because $SVRF \subseteq SVNF \subseteq SENF$, the VRF fault class can be considered "Stronger" than the VNF fault class, which in turn is stronger than the ENF fault class. He suggested that fault-based testing can be made more efficient by designing test generation algorithms to target the strongest fault class. The relationship between VRF, VNF and ENF fault classes implies immediately that not more than $n(n-1)$ tests are required to detect all the faults in any of these classes for an expression with $n$ distinct variables, or $m(n-1)$ tests for $m$ occurrences of $n$ variables. This is because each variable can be replaced by any of the other variables in a Variable Reference Fault. He suggested that the number of tests needed is much less, because of overlap between detection conditions.

2.4.2 Missing Condition Fault (MCF)

One of the most common implementation errors is the failure to validate input data or check preconditions. We will refer to this type of fault as a Missing Condition Fault. That is, a fault where the specification contains one or more conditions not implemented. Kuhn (1999) mentioned that Missing Condition Faults can be regarded as a special case of Variable Reference Fault.

Variable Reference Faults can now be divided into two. One, in which the variable substitution results in a Missing Condition Fault and the other in which one condition is replaced by another. The second type of fault could be described as an Incorrect Condition Fault, since the boolean variables typically represent some relation or condition in a specification.
Missing Condition Faults are extremely common (Marick 1990). Offutt et al (1997) used the following procedure, where predicates are assumed to be in DNF

- At the disjunctive level, where predicates are of the form $A \lor B \lor C \lor \ldots$, generate test values by holding all disjuncts but one false, then vary each one to be true in turn.
- At the conjunctive level, where predicates are of the form $A \land B \land C \land \ldots$, first find values that cause each clause to be true, then generate additional tests by holding all conjuncts but one true and vary each one to be false in turn.

As it turns out, this procedure is equivalent to generating Missing Condition Faults for each of the variables in the predicates being tested.

2.5 FAULT DETECTION CAPABILITY OF SPECIFICATION-BASED TESTING

Faults in boolean specifications constitute a hierarchy with respect to detectability. Several methods have been proposed for generating test cases for an implementation intended to satisfy a given specification, that is, a boolean formula. In those methods, typical faults are hypothesized and then test sets are derived to detect those faults. A variety of fault models have been proposed so far.

i. Variable Reference Fault
ii. Variable Negation Fault
iii. Expression Negation Fault

Let the boolean expressions $S$ and $S'$ denote a specification and its faulty implementation, respectively. A fault will be identified if and only if $S'$ evaluates to a different value than $S$. In other words if a test causes $S \oplus S'$ ($\oplus$ is exclusive – or) to
evaluate to true, then the fault will be detected. $S \oplus S'$ is referred to as the fault condition of the fault. Given a fault hypothesized, it is possible to generate a test that can detect the fault by computing its fault condition. Once a particular fault class is hypothesized, it is possible to generate test cases that can detect all faults in that class.

Tsuchiya et al (2002) quoted that Kuhn (1999) showed that there is a hierarchy of fault classes with respect to detection capability. Kuhn (1999) proposed a new fault class called Missing Condition Fault (MCF). A Missing Condition Fault refers to a fault where the specification contains a condition not implemented. (For example, $S = (p \land q) \lor r$, $S' = (q \lor r)$).

Kuhn (1999) suggested that the faults in the class should be hypothesized to generate efficient tests, since Missing Condition Faults are often equivalent to Variable Reference Faults, thus leading to higher detection capacity. Tsuchiya et al (2002) mentioned that this conclusion of Kuhn (1999) was too rushed, because the relationships between Missing Condition Faults and faults in other classes, including Variable Reference Faults, have not been sufficiently investigated.

Tsuchiya et al (2002) have analyzed the relationships between various fault classes in the hierarchy and extended it by including Missing Condition Faults in the hierarchy. He has presented a theorem showing the relationship between the test sets for Missing Condition Fault and Variable Reference Fault.

If the variable missed in $SMCF$ is the same variable negated in $SVNF$ then $SMCF \Rightarrow SVNF$.

Tsuchiya et al (2002) mentioned that it is guaranteed that tests that detect Missing Condition Faults will also detect Variable Negation Faults and Expression Negation Faults. However test that detects Missing Condition Faults may not be able to detect Variable Reference Faults and vice versa. He has noted that either tests for
Missing Condition Faults or those for Variable Reference Faults do not guarantee that faults in the other classes are detected.

In order to detect all the four types of faults, it suffices to use a test set that can detect Variable Reference Faults and Missing Condition Faults. They have concluded that in order to achieve high fault coverage, Variable Reference Faults should be hypothesized.

2.6 ON THE RELATIONSHIPS OF FAULTS FOR BOOLEAN SPECIFICATION-BASED TESTING

Lau et al (2001) proposed methods on the relationships of faults for boolean specification-based testing. Software testing aims at detecting software faults that are the result of human error during software development. Specification based testing derives test cases from the specification rather than the actual implementation. A fault based approach of generating test cases aim at detecting certain types of faults.

Lau et al (2001) mentioned that, according to Kuhn (1999), Variable Reference Fault resulted in every occurrence of a variable being replaced by another variable. They investigated whether findings of Kuhn (1999) regarding relationships among various fault models defined by him is applicable to the fault model defined by them. They also investigated what are the relationships among other types of faults so that more faults can be covered and fault detecting test cases can be selected in a more systematic and more efficient manner. They have studied the relationships among seven types of faults that are related to the operands in a boolean expression. A hierarchy showing the relationships of those faults summarizes the results. A test case that detects faults at the lower level of the hierarchy will always detect faults at the upper level of the hierarchy. They have taken seven faults for the analysis. They are Expression Negation Fault (ENF), Term Negation Fault (TNF), Literal Negation Fault
(LNF), Term Omission Fault (TOF), Literal Omission Fault (LOF), Literal Insertion Fault (LIF) and Literal Reference Fault (LRF).

2.7 GENERATING TEST DATA FROM SOFL SPECIFICATION

Offutt et al (1997) have developed a technique of generating test data from SOFL specifications.

Formal specifications represent a significant opportunity for testing because they precisely describe the functionality of the software in a form that can be easily manipulated by automated means. Offutt et al (1990) addresses the problem of developing formalizable, measurable criteria for generation of test cases from specifications. A model for generating SOFL specification with a case study is presented. They have analyzed the case study using mutation analysis and it was found that the technique was very effective and their results indicate that their technique can benefit software developers, who construct formal specifications during development.

Software functional specifications have been incorporated in several ways. They have been used as a basis for test case generation to check the output of the software on test input and as a basis for formalizing the test specification. A goal of the above is to produce mechanical procedures to derive test cases for formal specifications.

The primary concern for testers is how to produce the set of test cases that should be effective in finding faults in the program, adequate at providing some information about the quality of the program and preferably satisfy some requirements or criterions for testing that is repeatable and measurable. The expected output is produced from a test case with some knowledge of the specifications and the specifications are more likely to be formalized.
Predicate satisfaction uses precondition invariants and post condition invariants to create predicates and then generate test cases to satisfy individual clause within the predicates. Instead of covering just pre and post conditions it is important to relate the pre and post conditions with the test. Tests should also be used to find faults and also cover the input domain for SOFL specifications. It is important to test both S-modules and I-modules. Different approaches can be used to generate tests at different levels. A remaining problem is that of finding Missing Condition Faults. SOFL specifications are at two levels, the S-level module and the I-level module. Different approaches can be used to generate tests at different forms so that the goal of testing is different. Offutt et al (1990) adapts a system level technique for S-modules, a data flow based testing approach for integration testing among S-modules and extends the predicate based testing approaches for testing I-modules.

2.8 GENERATING TEST CASES FOR BOOLEAN SPECIFICATIONS

Boolean specifications represent the lowest level of condition process specification. A predicate-based approach is used to test the boolean specifications. Primarily the predicates are measured at three different levels, the disjunctive, conjunctive and the relational level. For convenience, it is assumed that the predicates are in Disjunctive Normal Form (DNF). The primary indent is that each clause in each predicate is tested independently. At the disjunctive level, the predicates are in the form \((A \lor B \lor C \lor \ldots)\). They are tested by holding all disjuncts but one false and varying each one to be true in turn. That is, tests are generated by finding values that satisfy the following partial truth table.

<table>
<thead>
<tr>
<th>Table 2.1 Test Generation Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A \lor B \lor C \lor \ldots )</td>
</tr>
<tr>
<td>( T )</td>
</tr>
<tr>
<td>( F )</td>
</tr>
<tr>
<td>( F )</td>
</tr>
</tbody>
</table>
These Test Cases sample from valid parts of the input space. In addition, the test engineer may include two or more disjuncts that are semantically meaningful because the program will be tested from the specification. It is only necessary to generate test data for input parameters and state variables in the pre and post conditions.

Thus, Offutt et al (1997) have provided an initial evaluation of the specification based test generation technique. They measured the quality using coverage criterion. Mutation testing is considered as one of the strongest testing technique and is a commonly used method for evaluating tests. The tests are evaluated based on their mutation score which is the ratio of the number of mutants killed over the total number of mutants. Experience has shown that mutation scores of over 90% are difficult to achieve and mutation scores over 95% are very difficult to achieve.

The Mothra mutation system was used and all mutants were generated for an implementation of Triang. Mothra created 842 non-equivalent mutants and the 31 test cases killed 817 mutants, for a mutation score of 97.03%.

2.9 TEST GENERATION FOR BOOLEAN EXPRESSIONS

Amit Paradkar et al (1995) proposed Test Generation methods for boolean expressions. They reviewed two test case generation techniques, and proposed an approach which combines Boolean Operator (BOR) Testing Strategy and Meaningful Impact Strategy so that the number of generated test cases is reduced. They have represented boolean expressions using Disjunctive Normal Form (DNF). This representation is used only for describing the Meaningful Impact Strategy. BOR testing strategy requires a set of tests that guarantee detection of boolean operator faults, including incorrect AND and OR operators and Missing/extra NOT operators (with the assumption that the expression does not contain faults of other types). The Meaningful Impact Strategy is applied to singular or non-singular expressions.
Combining MI-MIN strategy the number of test set is reduced. They have taken twenty specifications used by Weyuker et al (1994) for analysis. They have considered four types of faults:

i. Variable Negation Faults are those where every occurrence of a particular variable is replaced by its complement.

ii. Expression Negation Faults are those where an expression is replaced by its complement.

iii. Variable Replacement Faults are those where each occurrence of a particular variable is replaced by another variable from that expression.

iv. Operator Replacement Faults are those where a binary boolean operator (•, +) is replaced by another.

2.10 EFFECTS OF CHANGES IN FORMAL SPECIFICATIONS

Kuhn (1992) proposed a technique for analyzing the effects of changes in Formal Specifications. He defined the notion of predicate differences and showed how predicate differences may be used to analyze the effect of changes in formal specifications. Predicate differences may be used to define a meaning for the size of change to a formal specification.

2.11 APPROACHES TO SPECIFICATION-BASED TESTING

Richardson et al (1988) proposed approaches to Specification-Based Testing. They have described methods to select test cases from specifications. They extended the notions of error-based and fault-based testing to provide specification-Fault-Based testing. They also augmented the implementation-based techniques to actively use specifications as oracles in fault-based testing.
2.12 MUTATION OPERATORS FOR SPECIFICATION

Paul et al (2000) proposed mutation operation for specification. They have defined an extensive set of mutation operations for use with this method. They have presented the theoretical and experimental investigation of the relationships between the classes of faults detected by the various operators. They have recommended the set of mutation operators which yield good test coverage at a reduced cost compared to using all proposed operators. Each fault class has a corresponding mutation operator. Applying a mutation operator gives rise to a fault in that class. They have defined various mutation operators:

i. Operand Replacement Operator (ORO): Replace an operand, that is, a variable or constant, by another syntactically legal operand.

ii. Simple Expression Negation Operator (SNO): Replace a simple expression by its negation.

iii. Expression Negation Operator (ENO): Replace an expression by its negation.

iv. Logical Operator Replacement (LRO): Replace a logical operator (&, |, -) by another logical operator.

v. Relational Operator Replacement (RRO): Replace a relational operator (<, ≤, >, ≥, =, ≠) by any other relational operator, except its opposite.

vi. Missing Condition Operator (MCO): Delete Conditions (only simple expressions) from conjunction, disjunction, and implications.
vii. Stuck-At Operator (STO): This consists of two operators. Stuck-at-0, replace a simple expression with 0 and Stuck-at-1, replace a simple expression with 1.

viii. Associative Shift Operator (ASO): Change the association between variables.

Paul et al (2000) mentioned that the mutation operators generally do not correspond exactly to Kuhn's (1999) fault classes. If we combine ORO and RRO into a single operator, ORO+, this new operator generates a class of faults closely matching \( VRF \). A mutation operator is defined as one which generates a class of faults identical to \( VRF \). This Simple expression Replacement Operator (SRO) replaces a simple expression by every other syntactically valid simple expression. SRO sometimes generates higher order mutation, so by Woodward's (1993) principle, it should not be used for test generation.

ORO generates a large number of mutants, but provides the same set of test cases as all the operators combined consequently. It has 100% coverage. SNO, ENO and STO each provide second best coverage. SNO, however, generates significantly fewer mutants. MCO provides slightly less coverage while generating a smaller number of mutants. LRO generates a large number of mutants and provides good coverage. ASO has low coverage but generates very few mutants. They have concluded that ORO is sufficient to detect faults such as ORF, SNF and ENF.

2.13 SOFTWARE UNIT TEST COVERAGE AND ADEQUACY

Hong Zhu et al (1997) presented a survey on test coverage adequacy. Objective measurement of test quality is one of the key issues in software testing. Many test criteria have been proposed. The notion of adequacy criteria is examined together with in software dynamic testing.
Currently, software-testing literature contains two different, but closely related notions associated with the term test data adequacy criteria. First, stopping rule that determines whether sufficient testing has been done that it can be stopped. Second, test data adequacy criteria provide measurements of test quality. An adequacy criterion is an essential part of any testing method.

i. It can be an explicit specification for test case selection, such as a set of guidelines for the selection of test cases. Such a rule is usually referred to as a test case selection criterion.

ii. It is also in the form of specifying how to decide whether a given test set is adequate.

Mathematically speaking, test case selection criteria are generators, that is functions that produce a class of test sets from the program under test and the specification. Performing sufficient tests, the cost of testing is controlled by avoiding redundant and unnecessary tests, ensures the software quality. This role of adequacy criteria has been considered by some computer scientists (Weyuker 1986).

Specification-Fault-Based testing attempts to detect faults in the implementation that are derived from mis-interpreting the specification or the faults in the specification. Specification-Fault-Based testing involves planting faults in the specification. The program that implements the original specification is then executed and the results of the execution are checked against the original specification and those with planted faults. The adequacy of the test is determined according to whether all the planted faults are detected by checking. Ajei Gopal et al (1983) wrote about specification mutation in predicate calculus. They have identified a set of mutation operators that are applied to a specification in the form of pre or post conditions to generate mutants for the specification. Then the program under test is executed on the
test cases to obtain the output of the program. The input or output pair is needed to evaluate the mutants of the specification.

If a mutant is fortified by the test cases in the evaluation, it is killed by the test otherwise it remains alive. A mutant may remain alive for one of the following reasons. (i) The test data is inadequate. Mutation analysis can clearly reveal a weakness in program. (ii) The mutant is equivalent to the original program. The mutant and the original program always produce the same output, hence no test data can distinguish between the two. Normally, only a small percentage of mutants are equivalent to the original program.

2.14 TESTING BY SPECIFICATION MUTATION

Ajei Gopal et al (1983) noticed that some alterations to specifications were not useful as mutation operators, for example, replacing the various clauses in the specification by the truth values ‘true’ or ‘false’ tend to generate mutants that are trivial to kill and appear to be of little practical significance.

In an investigation of mutation operators for algebraic specifications, Woodward (1993) defined his set of mutation operators based on his analysis of errors in the specifications made by students. He considered algebraic specifications as term-rewriting systems. The original specification and the mutants of the specification are compiled into executable codes. When the execution of the original specification and a mutant on a given test case generate two different outputs, the mutant is regarded as dead. Otherwise, it is alive. In this way the test adequacy is measured without executing the program.
2.15 TRAFFIC ALERT AND COLLISION AVOIDANCE SUBSYSTEM (TCAS II)

The Safety Critical Systems Research Group at University of California produced a formal requirement specification for the Traffic Alert and Collision Avoidance Subsystem (TCAS II) required on all commercial aircrafts flying in U.S air space.

2.15.1 History of TCAS II System

Historically, the surveillance logic had been developed at MIT Lincoln Laboratory and the collision avoidance logic was developed at MITRE. Other organizations involved in the development activities include ARINC, the William J. Hughes Technical Center, as well as the avionics manufacturers (Allied Signal Honeywell and Rockwell-Collins).

2.15.2 Overview of TCAS II Operation

TCAS II is an airborne collision avoidance system that functions independently of the ground-based air traffic control system. TCAS II provides the crew of an aircraft with Traffic Advisories (TAs), which assist them in visually acquiring intruder aircraft, and Resolution Advisories (RAs), which provide them with recommended vertical escape maneuvers for avoidance of an intruder that becomes a threat (Bassam Abdul-Baki et al 1999).

The TCAS II system consists of four main subsystems: surveillance, the collision avoidance subsystem, displays and controls, and the monitor. Surveillance of the airspace around the TCAS-equipped aircraft is accomplished by making use of the Air Traffic Control Radar Beacon System (Mode A/C) and Mode S transponders that are already installed on all commercial and most general aviation and military aircraft, along with two TCAS antennas mounted on the top and bottom of the aircraft.
The CAS uses the information provided each second by the Surveillance Subsystem to determine the slant range and closing speed of the other aircraft in the vicinity of the TCAS aircraft and determines the time in seconds to Closest Point of Approach (CPA). When altitude reports of the other aircraft is available, the CAS logic tracks them and determines the other aircraft’s vertical speed. Using own aircraft altitude information, TCAS determines own aircraft altitude and altitude rate, and the relative altitude of each other aircraft. The tracking algorithm outputs (target range, range rate, relative altitude, and altitude rate) form the inputs for the TA, threat detection, and RA selection logic.

The other aircraft is determined to be proximate traffic, traffic (potentially triggering a TA) or a threat (potentially triggering an RA) based on whether it is within different protected volumes of airspace surrounding the TCAS aircraft. These volume thresholds are based on a range test, calculated using the time to CPA, and, for altitude reporting intruders, an altitude test calculated using the current or projected vertical separation at CPA.

If both aircrafts are equipped with TCAS and Mode S transponders, the Coordination Subsystem transmits RA information to the other TCAS-equipped aircraft via the Mode S data link to ensure that the maneuvers of the two aircrafts are complementary. For example, if the first aircraft to detect the threat condition selects a climb RA, the second aircraft will select a descend RA.

The controls and displays subsystem uses information provided by CAS to visually display proximate traffic, TAs and RAs to the cockpit crew. In addition, aural communication to the pilot is provided for TAs and RAs. Although TCAS is independent from the air traffic surveillance ground systems, air traffic controllers can nevertheless be notified of RAs in-progress through what is referred to as ‘RA downlink’, whereby TCAS devices transmit their RA status by means of the Mode S
ground interrogator. TCAS II Version 7 is a major change to TCAS II Version 6.04A, affecting every subsystem and all the functional areas of the CAS logic.

At the simplest level there are two essential stages to ensure the correctness of a piece of software. The first is to ensure that the required specifications are correct, and the latter is to ensure that the code completely meets the requirements specification.

2.16 CONCLUSION

Based on the information already available as mentioned above, further improvements and modifications have been proposed in the coming chapters.