Formalization of Software Testing Criteria using the Z Notation

Sergiy A. Vilkomir Jonathan P. Bowen
South Bank University, Centre for Applied Formal Methods
School of Computing, Information Systems and Mathematics
103 Borough Road, London SE1 0AA, UK
{*vilkoms, bowenjp*}*@sbu.ac.uk*
http://www.cafm.sbu.ac.uk/

Abstract

This paper describes an approach to formalization of criteria of computer systems software testing. A brief review of control-flow criteria is introduced. As a formal language for describing the criteria, the Z notation is selected. Z schemas are presented for definitions of the following criteria: statement coverage, decision coverage, condition coverage, decision/condition coverage, full predicate coverage, modified condition/decision coverage, and multiple condition coverage. This characterization could help in the correct understanding of different types of testing and also the correct application of a desired testing regime.

1 Introduction

Software testing criteria (or else, test data adequacy criteria or coverage criteria) play a large role of whole testing process. These criteria are used as [36]:

- stopping rules that determines whether sufficient testing has been done that it can be stopped;
- measurements of test quality when a degree of adequacy is associated with each test set;
- generators, for test data selection. Test sets are considered as equivalent if they satisfy the same criterion.

The use of testing criteria as regulatory requirements during software certification and licensing also has its own specific features and benefits. At the time of regulatory assessment, the stage of testing assessment is one of the most important where efforts of experts should be concentrated [33].

The methods and criteria of testing are traditionally divided into structural (or white-box) and functional (or black-box) aspects [21, 27]. Structural testing criteria, i.e. criteria which take into account an internal structure of the program, are in turn divided into data-flow and control-flow criteria, although the combination of the two has been considered [24, 31]. Data-flow criteria are based on the investigation of the ways in which values are associated with variables and how these associations can affect the execution of the program [36]. This group contains so-called *all-uses*, *all-defs*, *all-p-uses* and other criteria [14, 27]. Control-flow criteria examine logical expressions, which determine the branch and loop structure of the program. This group of criteria is considered in this paper.

The paper is structured as follows. Section 2 presents a brief review of control-flow criteria. Well-known criteria like statement coverage, decision coverage, condition coverage, decision/condition coverage, multiple-condition coverage, as well as relatively new criteria such as full predicate and modified condition/decision coverage (MC/DC) criteria are addressed.

In the scientific literature, criteria definitions are typically informal (in natural language). Sometimes these definitions are not clear enough and this can lead to inaccurate understanding. In section 3 of this paper, the task of producing formal criteria definitions is considered. As a formal language for describing the criteria, the Z notation is selected, which is used recently not only in academic surroundings [4] but also for industrial development of high-integrity systems such as safety-critical software [6, 7].

2 Review of control-flow criteria

2.1 Informal definitions

The most simple control-flow criteria are known from the 1960s and 1970s. The following are based on the well-known book by G. Myers [21]:

- statement coverage (SC): every statement in the program has been executed at least once;
- decision coverage (DC): every statement in the program has been executed at least once, and every deci-

sion in the program has taken all possible outcomes at least once;

- condition coverage (CC): every statement in the program has been executed at least once, and every condition in each decision has taken all possible outcomes at least once;
- decision/condition coverage (D/CC): every statement in the program has been executed at least once, every decision in the program has taken all possible outcomes at least once, and every condition in each decision has taken all possible outcomes at least once;
- multiple condition coverage (MCC): every statement in the program has been executed at least once, and all possible combinations of condition outcomes in each decision have been invoked at least once.

These definitions use concepts of 'decision' and 'condition'. A decision is a program point at which the control flow can divide into various paths. An example of a decision is the IF-THEN-ELSE construction in Pascal and other imperative programming languages. A decision is a Boolean expression consisting of a one or several condition combined by logical connectives. A condition is an elementary Boolean expression (atomic predicate), which cannot be divided into further Boolean expressions.

All the mentioned definitions involve the statement coverage criterion as a component part. This inclusion is slightly artificial because pure decision or condition coverage is not connected with statement coverage. The purpose of this inclusion is to establish the following partial ordering of the control flow criteria: criterion *A* is stronger (subsumes) criterion *B* if every test set that satisfies *A* also satisfies *B*. Other relations between testing criteria have been also considered [13].

The multiple condition coverage criterion is the strongest and requires full searching of various combinations of conditions values. However, an excess of test cases can be required. If the number of conditions in a decision is equal to *n*, then the number of test cases to satisfy this criterion is 2^n; this is not normally possible in practice even for relatively moderate values of *n*. The other mentioned criteria are weaker and require considerably less test patterns. Thus, condition coverage requires two tests for each condition and the total quantity of tests equals $2n$. However, in this connection, testing of combinations of conditions values is missing and such testing volume is not sufficient for safety-critical software [12].

An intermediate position between multiple condition coverage criterion and other criteria is taken up by modified condition/decision coverage (MC/DC) criterion:

- MC/DC: every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken on all possible outcomes at least once, and each condition has been shown to independently affect the decision's outcome. A condition is shown to independently affect a decision's outcome by varying just that condition while holding fixed all other possible conditions [28].

As evident from the definition, MC/DC criterion requires achievement of statement coverage and condition coverage criteria combined with the requirement that each condition should affect the decision's outcome. It means that the outcome of a decision changes as a result of changing a single condition [11]. This criterion requires testing of various (but not all) combinations of conditions values but the number of tests still grows by linear law.

Full predicate coverage criterion [23] is similar to MC/DC but is weaker [1]. This criterion is based on the specifications and the original definition uses different terms (for example, 'clause' instead 'condition'). But it is possible to reformulate it for uniformity with previously defined criteria and using a definition from [23] as a base:

- full predicate coverage (FPC): every statement in the program has been executed at least once, and each condition in a decision has taken all possible outcomes where the value of a decision is directly correlated with the value of a condition. This means, a decision changes when a condition changes.

Thereby, the distinction between full predicate coverage and MC/DC is that while varying a condition, it is not necessary to fix all other possible conditions. In this case it is easier to assemble test cases than for MC/DC. But on the other hand, the effect of varying a condition on a decision can be masked by the other conditions.

In addition to those mentioned above, other control-flow criteria has been introduced (for example, LCSAJ [35], DD-path [25], and Object Level [15] coverage criteria). Diagrams, which show interrelation between control-flow and data-flow criteria are considered in [15, 22, 26, 36].

2.2 An example

The following example illustrates the contents of the testing criteria for a decision, containing three conditions and implementing the principle of the logical majorization '2 from 3': the decision is true if and only if any two conditions are true. This principle is widely used for data processing in many safety-critical systems. Note, that this example of the use of the testing criteria is given only for simplification of explanation and has no a practical sense because a number of the conditions is small and full searching of all test cases is easy achieved.

Let *d* be the decision and *A*, *B*, and *C* be conditions. Then

$$d = (A \wedge B) \vee (A \wedge C) \vee (B \wedge C)$$

The values of conditions are 1 (TRUE) or 0 (FALSE). Eight combinations of the values of the conditions exist.

For DC, CC, D/CC, and FPC, two combinations are sufficient. For MCC, the set containing all eight combinations is required. The combinations, which satisfy the MC/DC criterion, are given in Table 1. The pair, suited for each condition, is marked '$*$'.

combination number	values				variations			MC/DC
	A	*B*	*C*	*d*	*A*	*B*	*C*	
1	1	1	1	1				
2	1	1	0	1	$*$	$*$		+
3	1	0	1	1			$*$	+
4	0	1	1	1				
5	1	0	0	0		$*$	$*$	+
6	0	1	0	0	$*$			+
7	0	0	1	0				
8	0	0	0	0				

Table 1. Combinations satisfied the MC/DC criterion

The subset of four combinations, marked '+', satisfies the MC/DC criterion.

A more detailed example is addressed in a technical report [32]. This example takes into account the following factors:

- dependence of the values of the conditions and decisions on input data;
- dependence of the specific decision on its place in the computer program, i.e., on the values of other decisions in the program;
- dependence of the conditions in the specific decision on each other, i.e., the possibility of one condition takes its value depending on the value of other conditions in this decision.

3 Formalization of control-flow criteria in Z

The criteria such as full predicate coverage and modified condition/decision coverage are quite complicated and require additional explanation. Various understandings of these definitions are possible giving rise to ambiguity. To help alleviate this situation, the elaboration of formalized definitions of testing criteria, which describe criteria contents in rigorous mathematical form, is presented here.

Formalization of certain criteria has been carried out using set theory [13], graph theory [24], predicate logic [20, 31], temporal logic [2]. In this paper, the Z notation [29] is used for the formal definition of the criteria.

The reasons for choosing the Z notation are follows. Z has been used for a number of digital systems in a variety of ways to improve the specification of computer-based systems [3]. Many textbooks on Z are now available (e.g., [18]). The teaching of Z has become of increasing interest [5]. Concerning software testing, the Z notation has been used to derive tests from model-based specifications [10, 30], for the testing of abstract data types (modules, classes, package, clusters) [16], automatic test case generation [8, 9], the selection of test cases and evaluation of test results [17]. So, when Z is used for software development and testing, it is expedient to use testing criteria, also formulated in Z.

3.1 Basic concepts

The following two given sets are used further for definitions of testing criteria:

$$[INPUT, STATEMENT]$$

The first set is a set of possible values of input program variables. So, any $i \in INPUT$ is a vector whose components are specific values of all input quantities. The second set is a set of all program statements. The further specification of the nature of these sets is not necessary for the creation of the criteria definitions.

When the specific values of input variables are determined and the program is executed, usually only a part of program branches are involved, i.e. only a part of program statements are activated. Below *path i* is a set of program statements that are executed when the input variables value is *i*.

$$path : INPUT \rightarrow \mathbb{P}\, STATEMENT$$

The set *Bool*, used for the determination of the value of conditions, contains only two elements – 0 and 1.

$$Bool == \{0, 1\}$$

PartInput is a set of all non-empty subsets of *INPUT*, excluding the set *INPUT*.

$$PartInput == \mathbb{P}_1\, INPUT \setminus \{INPUT\}$$

The abbreviation *cond* is introduced for the set of all possible conditions, which are considered as logical predicates on *INPUT*, i.e. for any $i \in INPUT$ the value of a condition equals 0 (FALSE) or 1 (TRUE). The set of conditions will later be restricted and conditions linked with each specific decision will be used.

$$cond == INPUT \rightarrow Bool$$

The following schema[1] describes the notion of 'decision'. On the one hand, each decision is a statement of

[1] All schemas in this paper were checked using the ZTC type-checker package [19].

a program (variable *decst*). On the other hand, each decision is a logical predicate (the function *value*) on the subset *decinput* of input data, for which the executed path passes through this decision, i.e. $decst \in path\ i$. Correspondingly, $decinput0$ and $decinput1$ are the sets, where the function *value* (i.e., the value of the decision) equals 0 and 1 respectively. These sets partition the set *decinput*, i.e. their union equals *decinput*, and their intersection is the empty set. From the definition of *PartInput*, both of them are non-empty. Thereby, the decisions, which are either all 0 or all 1, are excluded from consideration. The reasons are that all such decisions are covered when the statement coverage criterion is satisfied and that 'affect the decisions outcome' (see above the definition of the MC/DC) and 'the correlation with the value of a condition' (see above the definition of the full predicate coverage) are impossible for such degenerate decisions.

dec

$$
\begin{array}{l}
decst : STATEMENT \\
decinput : \mathbb{P}_1\ INPUT \\
decinput0, decinput1 : PartInput \\
argdec : \mathbb{P}_1\ cond \\
value : cond \\
\hline
decinput = \{i : INPUT \mid decst \in path\ i\} \\
\langle decinput0, decinput1 \rangle \text{ partitions } decinput \\
argdec = \{c : cond \mid \mathrm{dom}\, c = decinput \land \\
\quad \mathrm{ran}(decinput \lhd c) = Bool\} \\
\mathrm{dom}\, value = decinput \\
decinput1 = \{i : INPUT \mid value\ i = 1\}
\end{array}
$$

The set *argdec* contains all conditions that are components of a given decision. Each conditions should takes both 0 and 1. This means that conditions that are either all 0 or all 1 are not considered. The reasons are the same as for decisions – each such condition is always covered when a decision is covered and varying condition's value (for MC/DC and full predicate coverage) is impossible for such degenerate conditions.

To describe the process of testing, the set of testing data is introduced. This set, named *testset*, can satisfy or not satisfy to testing criteria.

3.2 Formal definitions

The following schema gives the definition when testing data satisfy the statement coverage criterion.

StatementCoverage

$$
\begin{array}{l}
testset : \mathbb{P}_1\ INPUT \\
\hline
\forall v : STATEMENT \bullet (\exists i : testset \bullet v \in path\ i)
\end{array}
$$

As the statement coverage criterion is a component part of all other criteria, its schema is in the signature of all other schemas used for defining testing criteria. Thus, the set *testset* is used (via the *StatementCoverage* schema) for definitions all testing criteria.

The following schema determines a formal definition of the decision coverage criterion based on the fact that for any specific decision *d* the set *testset* should overlap with both $d.decinput0$ and $d.decinput1$. In this case, the given decision takes both the value 0 (for the testing data from $d.decinput0$) and the value 1 (for the testing data from $d.decinput1$).

DecisionCoverage

$$
\begin{array}{l}
StatementCoverage \\
\hline
\forall d : dec \bullet (testset \cap d.decinput0 \neq \varnothing) \land \\
\quad (testset \cap d.decinput1 \neq \varnothing)
\end{array}
$$

The following schema determines the condition coverage criterion and is analogous with the previous schema. It claims that a pair of input data from testing set should exist, for which the condition takes different values (both 0 and 1).

ConditionCoverage

$$
\begin{array}{l}
StatementCoverage \\
\hline
\forall d : dec;\ c : cond \mid c \in d.argdec \bullet \\
\quad (\exists i_0, i_1 : testset \cap d.decinput \bullet c\ i_0 \neq c\ i_1)
\end{array}
$$

The formal description of the decision/condition coverage criterion uses the fact that this criterion is the union of the decision criterion and the condition criterion. So, the schema of this criterion contains only references to two previous schemas.

DecisionConditionCoverage

$$
\begin{array}{l}
DecisionCoverage \\
ConditionCoverage
\end{array}
$$

The requirement, which determines the full predicate coverage criterion, is similar to the corresponding requirements for the condition coverage: a testing set should contain such input data (i_0 and i_1) that the value of a condition equals 0 for one of them and equals 1 for the other.

FullPredicateCoverage

$$
\begin{array}{l}
StatementCoverage \\
\hline
\forall d : dec;\ c : cond \mid c \in d.argdec \bullet \\
\quad (\exists i_0, i_1 : testset \bullet (d.value\ i_0 \neq d.value\ i_1) \land \\
\quad\ (c\ i_0 \neq c\ i_1))
\end{array}
$$

But if in case of condition coverage there are no restrictions on the input data (except their membership in $d.argdec$; i.e., for these input data the decision, as a program statement, should be executed), then for full predicate coverage there is the additional restriction: for these input data the decision also should have different values, i.e., both the condition and the decision should vary simultaneously.

The following schema determines the MC/DC criterion.

ModifiedConditionDecisionCoverage

FullPredicateCoverage

$$\begin{aligned}
&\forall\, d : dec;\ c : cond \mid c \in d.argdec \bullet \\
&\quad (\exists\, pair : \mathbb{P}_1 (d.decinput0 \times d.decinput1) \bullet \\
&\qquad (\forall\, i_0, i_1 : INPUT \mid (i_0, i_1) \in pair \bullet \\
&\qquad\quad (c\ i_0 \neq c\ i_1 \wedge \\
&\qquad\quad (\forall\, othercond : d.argdec \mid othercond \neq c \bullet \\
&\qquad\qquad othercond\ i_0 = othercond\ i_1)))) \Rightarrow \\
&\quad (\exists\, j_0, j_1 : testset \bullet (j_0, j_1) \in pair)
\end{aligned}$$

As the MC/DC criterion subsumes the full predicate criterion, this schema contains the *FullPredicateCoverage* schema, combined with an additional restriction. This restriction requires that, in case varying of one condition, all other conditions (*othercond*), which make up the given decision, should not vary, if it is possible in principle. It means that the values of each *othercond* for i_0 and i_1 are equal.

The subset of pairs of input data, which display this combination, is denoted as *pair*. If such a non-empty subset *pair* exists, then at least one from its elements should be a member of the testing set.

This formal definition eliminates a certain shortcoming of the definition in natural language. The original definition does not clearly describe the situation when, varying one condition, it is impossible to fix all other conditions. It is not necessarily the best option to consider that the MC/DC is not satisfied in this situation. The formal definition shows that testing data should answer the full predicate coverage criterion for this condition. It means that it is acceptable to vary this condition simultaneously with the decision but without fixing the value of all other conditions.

The last scheme determines the multiple condition coverage criterion.

MultipleConditionCoverage

StatementCoverage

$$\begin{aligned}
&\forall\, d : dec;\ condset : \mathbb{P}\, cond \mid \\
&\quad condset \in \mathbb{P}\, d.argdec \bullet \\
&\quad (\exists\, comb : \mathbb{P}_1\, d.decinput \bullet (\forall\, i : comb \bullet \\
&\qquad (\forall\, c : condset \bullet c\ i = 1) \wedge \\
&\qquad (\forall\, c : d.argdec \mid c \notin condset \bullet c\ i = 0))) \Rightarrow \\
&\quad (testset \cap comb \neq \varnothing)
\end{aligned}$$

The definition claims that the testing data set (*testset*) should contain the data for testing every combination of the values of conditions into a decision, if such combination is possible in principle (i.e., if there are input data for which the value of conditions make up the given combination). In the schema given above, each combination of the values of the conditions is clearly defined by the subset *condset* of the conditions, which equal 1 for this combination. Accordingly, the other conditions from $d.argdec$, which are not members of *condset*, equals 0 for this combination.

The subset of input data, which display this combination, is denoted as *comb*. If such a non-empty subset *comb* exists, then at least one from its elements should be a member of the testing set.

4 Conclusions and future work

The subject of this paper is the formalization of criteria for complex computer systems software testing. Control-flow criteria, i.e., criteria using logical expressions, which determine the branch and loop structure of the program, are considered. This group includes well-known criteria [21] and relatively new criteria – full predicate [23] and modified condition/decision coverage (MC/DC) [28] criteria. A brief review and examples illustrating the testing criteria are introduced.

The Z notation [3, 29] is used for the formal definition of the criteria. Z schemas formally describing all main control-flow testing criteria are presented. These definitions help to eliminate the possibility of inaccurate understanding, which is likely for definitions in natural language. In particular, the MC/DC formal definition takes into consideration the situation when, varying a condition, it is impossible to fix all other conditions because of mutual dependence of the conditions and decisions.

The formal definitions could be used, in particular, for laying down regulatory requirements [34] for the testing of safety-critical software. The proposed approach could also be used for the formalization of other testing criteria (e.g., data-flow control criteria). Another direction for future work could be using formal definitions for detailed analysis of the content and applicability of the most complicated existing criteria and for producing new criteria. One of the possible criteria for further analysis is the modified condition/decision coverage criterion.

References

[1] Abdurazik, A., Amman, P., Ding, W., Offutt, J. Evaluation of Three Spesification-based Testing Criteria. *Sixth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS2000)*, Tokyo, Japan, September 2000.

[2] Ammann, P. E., Black, P. E. Test generation and recognition with formal methods. *The First International Workshop on Automated Program Analysis, Testing and Verification*, Limerick, Ireland (June 2000).

[3] Bowen, J. P. *Formal Specification and Documentation Using Z: A Case Study Approach*. International Thomson Computer Press, 1996.

[4] Bowen, J. P. Z: A Formal Specification Notation. In: Frappier, M., Habrias, H., editors. *Software Specification Methods: An Overview Using a Case Study*, Springer-Verlag, 2001, Chap. 1, pp. 3–19.

[5] Bowen, J. P. Experience Teaching Z with Tool and Web Support. *ACM SIGSOFT Software Engineering Notes*, Vol. 26, No. 2, March 2001, pp. 69–75. Also available as Technical Report SBU-CISM-00-30, SCISM, South Bank University, London, UK, 2000.

[6] Bowen, J. P., Hinchey M., G. The Use of Industrial-Strength Formal Methods. *Proceedings of 21st International Computer Software and Application Conference (COMPSAC'97)*, Washington D.C., USA, 13–15 August 1997. IEEE Computer Society Press, 1997, pp. 332–337.

[7] Bowen, J. P., Hinchey, M. G., editors. *Industrial-Strength Formal Methods in Practice*. Springer-Verlag, FACIT series, 1999.

[8] Burton, S., Clark, J., Galloway, A., McDermid, J. Automated V&V for high integrity systems, a target formal methods approach. *Proceedings of the 5th NASA Langley Formal Methods Workshop*, June 2000.

[9] Burton, S., Clark, J., McDermid, J. Testing, Proof and Automation. An Integrated Approach. *The Proceedings of the 1st International Workshop of Automated Program Analysis, Testing and Verification*, June 2000.

[10] Carrington, D., Stocks, P. A tale of two paradigms: Formal methods and software testing. In: J. P. Bowen and J. A. Hall, editors. *Z User Workshop, Cambridge, 1994*, Workshop in Computing. Springer-Verlag, 1994. Proceedings of the Eighth Annual Z User Meeting, pp. 51–68.

[11] Chilenski, J., Miller, S. Applicability of modified condition/decision coverage to software testing. *Software Engineering Journal*, September 1994, pp. 193–200.

[12] Dupuy, A., Leveson, N. An Empirical Evaluation of the MC/DC Coverage Criterion on the HETE-2 Satellite Software. *Proceedings of the Digital Aviation Systems Conference (DASC)*, Philadelphia, October 2000.

[13] Frankl, P. G., Weyuker, E. J. A Formal Analysis of the Fault-Detecting Ability of Testing Methods. *IEEE Transactions on Software Engineering*, Vol. 19, No. 3, March 1993, pp. 202–213.

[14] Frankl, P. G., Weyuker E. J. An Applicable Family of Data Flow Testing. *IEEE Transactions on Software Engineering*, Vol. 14, No. 10, October 1988, pp. 1483–1497.

[15] Haworth, B. Adequacy criteria for object testing. *Proceedings of the 2nd International Software Quality Week Europe 1998*, Brussels, Belgium, November 1998.

[16] Hayes, I. J. Specification Directed Module Testing. *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 1, January 1986, pp. 124–133.

[17] Hörcher, H-M. Improving Software Tests using Z Specifications. In: Bowen, J. P., Hinchey, M. G., editors. *ZUM'95: The Formal Specification Notation*. Springer-Verlag, Lecture Notes in Computer Science, Vol. 967, 1995, pp. 152–166.

[18] Jacky, J. *The Way of Z: Practical Programming with formal Methods*. Cambridge University Press, 1997.

[19] Jia, X. *ZTC: A Type Checker for Z Notation. User's Guide. Version 2.03, August 1998*. Division of Software Engineering, School of Computer Science, Telecommunication, and Information Systems, DePaul University, USA, 1998.

[20] Kaufman, A. V., Chernonozhkin, S. K. Testing Criteria and a System for Evaluation of the Completeness of a Test Set. *Programming and Computer Software*, No. 6, 1998, pp. 301–311.

[21] Myers, G. *The Art of Software Testing*. Wiley-Interscience, 1979.

[22] Ntafos, S. A Comparison of Some Structural Testing Strategies. *IEEE Transactions on Software Engineering*, Vol. 14, No. 6, June 1988, pp. 868–874.

[23] Offutt, A. J., Xiong, Y., Liu, S. Criteria for Generating Specification-Based Tests. *Proceedings of Fifth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'99)*, Las Vegas, Nevada, USA, October 18–21, 1999, pp. 119–129.

[24] Podgurski, P., Clarke, L. A Formal Model of Program Dependences and Its Implications for Software Testing, Debugging and Maintenance. *IEEE Transactions on Software Engineering*, Vol. 16, No. 9, September 1990, pp. 965–979.

[25] Prather, R. E. Theory of Program Testing – An Overview. *Bell System Technical Journal*, Vol. 62, No. 10, 1984, pp. 3073–3105.

[26] Rapps, S., Weyuker, E. J. Selecting Software Test Data Using Data Flow Information. *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 4, April 1985, pp. 367–375.

[27] Roper, M. *Software Testing*. McGraw-Hill, 1994.

[28] RTCA/DO-178B. *Software Considerations in Airborne Systems and Equipment Certification*, RTCA, Washington D.C., USA, 1992.

[29] Spivey, J. M. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science, 2nd edition, 1992.

[30] Stocks, P., Carrington, D. A Framework for Specification-Based Testing. *IEEE Transactions on Software Engineering*, Vol. 22, No. 11, November 1996, pp. 777–793.

[31] Tai, K-C. Theory of Fault-Based Predicate Testing for Computer Programs. *IEEE Transactions on Software Engineering*, Vol. 22, No. 8, August 1996, pp. 552–562.

[32] Vilkomir, S. A., Bowen, J. P. Formalization of Control-flow Criteria of Software Testing , *Technical Report SBU-CISM-01-01, SCISM , South Bank University*, London, UK, January 2001.

[33] Vilkomir, S. A., Kharchenko, V. S. An 'Asymmetric' Approach to the Assessment of Safety-Critical Software During Certification and Licensing. *Project Control: the Human Factor, Proceedings of ESCOM–SCOPE 2000 Conference*, 18–20 April 2000, Munich, Germany, pp. 467–475.

[34] Vilkomir, S. A., Kharchenko, V. S. Methodology of the review of software for safety important systems. *Safety and Reliability. Proceedings of ESREL'99 – The Tenth European Conference on Safety and Reliability*, Munich-Garching, Germany, 13–17 September 1999, Vol. 1, pp. 593–596.

[35] Woodward, M. R., Hedley, D., Hennell, M. A. Experience with Path Analysis and Testing of Programs. *IEEE Transactions on Software Engineering*, Vol. SE-6, No. 3, May 1980, pp. 278–286.

[36] Zhu, H., Hall P. A., May, H. R. Software unit test coverage and adequacy. *ACM Computing Surveys*, Vol. 29, No. 4, December 1997, pp. 336–427.