JOURNAL OF ELECTRONIC TESTING: Theory and Applications 17, 175–183, 2001

Constraint Based Criteria: An Approach for Test Case Selection in the Structural Testing

SILVIA REGINA VERGILIO
UFPR—Curitiba CP: 19081, 81531-970, Brazil
silvia@inf.ufpr.br

JOSÉ CARLOS MALDONADO
ICMSC-USP—São Carlos, 13560-970, Brazil
jcmaldon@icmc.sc.usp.br

MARIO JINO
DCA-FEEC-UNICAMP—Campinas, CP: 6101, 13083-970, Brazil
jino@dca.fee.unicamp.br

Received October 6, 2000; Revised February 28, 2001

Editors: V.H. Champac and M. Lubaszewski

Abstract. The selection of test cases to satisfy a structural testing criterion is a very important task because it concerns the quality of the generated test cases. The question is "How to select a test case or path to cover an element required by a structural criterion?" The Constraint Based Criteria are proposed with the goal of answering this question and improving the efficacy, that is, the number of revealed faults. These criteria permit the use of different testing strategies by associating constraints to the required elements. The constraints describe faults generally not revealed by the structural technique. The Constraint Based Criteria can also be used to assess test data sets adequacy.

Keywords: structural testing criteria, constraint based testing, mutation analysis

1. Introduction

Testing is a fundamental activity for software quality assurance. The main goal of testing is to reveal faults. Thus, a good test case is one that has a high probability of finding an unrevealed fault. Testing techniques have been proposed to reveal a great number of faults with minimal effort and costs. There are three groups of techniques: functional technique—uses the functional specification of a program to derive test cases; structural technique—derives test cases based on paths in the control flow graph of a program; and fault-based technique—derives test cases to show the presence or absence of typical faults in a program. These techniques are generally associated to testing criteria which are predicates used to decide on the end of the testing activity, that is, to consider a program tested enough and to help the tester in the task of test cases selection.

This work focuses on structural testing criteria. These criteria generally establish elements to be exercised by the execution of a set of paths in the program. Structural testing criteria divide the input domain of the program being tested into sub-domains and they generally require the execution with at least one point in each sub-domain corresponding to a required element. Once the input domain is divided, the question is "Which points from each sub-domain should be selected?" [6] Answer: Points with the highest probability of revealing a fault. This question concerns the generation of test cases to satisfy a criterion, which is very complex to be automated since there is no general algorithm to determine a set of test cases that satisfies a given criterion. It is not possible to determine even that such set exists [5].

There are many test data generation techniques using different principles for choosing points from the program input domain, associated to certain types of faults: Domain Testing [25], Boundary Values Analysis [12], Constraint Based Testing [3], BOR/BRO-Boolean and Relational Operator Testing [15, 16]. These techniques are not usually used to satisfy structural testing; however, this use is supposed to improve the efficacy of the generated test cases, that is, to increase the number of revealed faults. The "Constraint Based Criteria" are proposed with the goal of answering the question above and of increasing the efficacy of the test data generated to satisfy structural testing criteria. They permit the use of structural testing with the diverse principles of the test data generation techniques. The Constraint Based Criteria (CBC) can also be used to determine coverage measures and to evaluate data test sets.

The CBC proposal is the result from a study with the three testing techniques above [19] and several experiments carried out with structural testing criteria [10, 21]. The constraint based criteria applicability has been shown in a previous experiment described in [22]. This experiment used simple programs and the generation of the required elements as well the adequacy analysis were done manually. To explore the CBC's applicability with more complex programs we extended the testing tool POKE-TOOL [18] to support CBC. An instrumentation model for constraint based criteria was established and an evaluation module, based on this model, was developed. Results are presented here from another experiment using the implemented module.

In Section 2, we present an overview of testing criteria with the goal of introducing the concepts, the criteria and tools. Section 3 introduces the CBC: the main purposes and definition. Section 4 describes the experiment with the POKE-TOOL. The results presented in this section show the applicability of the CBC and an increase in the number of revealed faults. Section 5 presents the conclusions and future work.

2. An Overview of Testing Criteria

Testing criteria have the goal of selecting "good test cases". They are predicates to be satisfied and are usually derived applying different testing techniques: functional, structural, and fault-based techniques. They require program elements to be tested and provide coverage metrics. In general, structural criteria require the execution of paths in the program. These paths must exercise elements from the source code or from the control flow graph G of the program being tested. They are Control Flow Based or Data Flow Based Criteria. Control flow based criteria require exercising the nodes, edges and paths of the control flow graph. Data flow based criteria require associations between definitions and uses of variables in a program [8, 10, 11, 13, 17]. The structural criteria described below illustrate the ideas presented in this paper.

Control Flow Based Criteria:

- All-nodes: every node of G must be executed at least once during testing.
- All-edges: every edge of G must be traversed at least once during testing.
- All-paths: require the execution of all paths of G.

Data Flow Based Criteria:

- All-uses: it requires the execution of paths in G that cover all associations. An association (i, j, x) or $(i, (j, k), x)$ is established if a variable x is defined in a node i, x is used in a node j or is used in a predicate associated to edge (j, k), and there is a path from node i to j (or edge (j, k)) that does not redefine x. All-defs and all-du-paths are examples of other criteria from this family [13].
- All-potential-uses: it requires the execution of paths in G that cover all potential-associations. A potential-association (i, j, x) or $(i, (j, k), x)$ is established if a variable x is defined in a node i and there is a path from node i to j (or edge (j, k)) that does not redefine x. All-potential-uses/du and all-potential-du-paths are examples of other criteria from this family [10].

Several tools have been developed to support the application of data flow based criteria [1, 5, 7]. We use POKE-TOOL [1], a multi-language tool that supports the control flow based criteria—all-edges and all-nodes, and the "Potential Uses Criteria"—all-potential-uses (PU), all-potential-uses/du, and all-potential-du-paths. A criterion C_1 includes a criterion C_2 (Notation: $C_1 \rightarrow C_2$) if, for every program, every test data set which satisfies C_1 also satisfies C_2. If neither C_1 includes C_2 nor C_2 includes C_1, C_1 and C_2 are incomparable. Aspects of the inclusion relation among these criteria are described in [9, 13]. The data flow based criteria are stronger than control flow based criteria, i.e., they include control flow based criteria.

Fault-based criteria require the program execution with test data describing faults with the goal of proving the presence or absence of the faults in the program being tested. Mutation Analysis is a fault-based criterion. It establishes mutant programs which are versions of the original program being tested. They differ from the original program by just a simple modification. Test cases must be derived with the goal of killing mutants. A mutant will be considered dead if its behavior concerning a test case is different from that of the original program. Since a mutant is represented by a single mutation in a statement, the state of the mutant program must differ from the original program state after the mutated statement. This fact is named the necessity condition, because it may not be sufficient to kill the mutant. Another condition, the sufficiency condition, guarantees that the incorrect state will cause an incorrect output, as desired. However, determining sufficient conditions is an undecidable problem.

DeMillo and Offut [3] proposed a technique named Constraint Based Testing (CBT) that uses algebraic constraints to generate test cases to kill mutants. A system named Godzila was implemented to apply the CBT technique [4]. Godzila generates test data with the goal of killing the mutants generated by Mothra (a tool based on Mutation Analysis). Necessity constraints describing faults associated with the Mothra operators are generated and testing cases satisfying these constraints are derived. DeMillo and Offut assume that a test case meeting the necessity condition will usually satisfy the sufficiency condition, contributing to kill one or more mutants.

The Constraint Based Criteria, defined in Section 3, are strongly based on DeMillo and Offut's work. Another tool that supports Mutation Analysis is Proteum [2]. Using Proteum and POKE-TOOL we have conducted many empirical studies to compare testing criteria and different testing techniques. These studies are based on efficacy, number of test cases and strength [9, 14, 21]. We have also studied some strategies for generation of test cases to satisfy testing criteria [20]. These theoretical and practical studies include Domain Testing [25], Constraint Based Testing [3], BOR/BRO-Boolean and Relational Operator Testing [15]. A couple of interesting results, presented in the next section, motivated us to propose the Constraint Based Criteria.

3. Constraint Based Criteria

Different strategies to satisfy structural testing criteria were used in the experiments; we can summarize the results which constitute strong motivations for defining the Constraint Based Criteria, as follows:

1. Necessary conditions were not always sufficient to reveal faults, specially in the case of computational faults;
2. Determining sufficient conditions for computational faults was more difficult than determining sufficient conditions for domain faults;
3. In many cases the sufficient condition was a condition to execute a particular path in the program;
4. The stronger the path based criteria the greater the probability of selecting a set of paths containing a path that can reveal the fault.

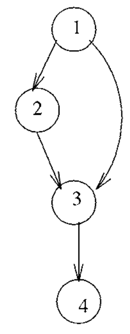
Program *max* Fig. 1 [3] that prints the largest of its inputs m and n exemplifies the observations. To reveal the fault associated to node 1, replacement of m by $abs(m)$, the necessity condition is $max <> abs(m)$; however, this condition is not sufficient. The sufficiency condition to reveal the fault is $n < m$, that is, the condition to execute path 1 3 4. In many cases, the combination of necessity conditions and paths' conditions has a high probability of meeting sufficiency conditions and of revealing the fault. Applying the Constraint Based Technique with structural criteria can yield good results. Points that satisfy these criteria may describe common faults and have a high probability of revealing them. The selection of more than one test case satisfying necessity conditions is possible and a path can be executed by different test cases.

3.1. Definition of Constraint Based Criteria

The Constraint Based Criteria are defined by associating constraints to an element required by a structural testing criterion. Thus, the Constraint Based Criteria require pairs (E,C) (structural element, constraint), named constrained elements. A constrained element will be covered if C is satisfied during the execution of

```
     void max()
   { float m,max,n;
1    scanf{"%f", &m);
1    scanf("%f ", &n);
1    max = m; --> max = abs(m);
1    if (n>=m)
2       max = n;
3    printf("%f", max);
4  }
```

a) Source Code



b) Program Graph

Required Elements	Paths
(1,(1,3),{m,max,n})	1 3 4
(1,(1,2),{m,max,n})	1 2 3 4
(2,(-,-),{max})	

c) Elements Required by All-Potential Uses Criterion

Fig. 1. Program *max*.

a path that covers E. We can use any structural testing criterion to derive a Constraint Based Criterion. The Constraint Based Criteria using the best known structural testing criteria are presented below:

- Constrained Control Flow Based Criteria: all-constrained-nodes criterion and all-constrained-edges criterion are defined. For these criteria constrained elements are, respectively, constrained-nodes and constrained-edges, and they are covered if a path that covers the correspondent node or edge is executed and C is satisfied during this execution.
- Constrained Data Flow Based Criteria: all-constrained-uses and all-constrained-potential-uses criteria are defined. The elements required by them are constrained associations or constrained potential-associations.
  - All-constrained-uses: a constrained association is given by $((i, k, x), C)$ or $((i, (j, k), x), C)$. It will be covered if a path that covers the association is executed and C is satisfied during this execution.
  - All-constrained-potential-uses: a constrained potential-association is given by $((i, k, x), C)$ or $((i, (j, k), x), C)$. It will be covered if a path that covers the potential-use association is executed and C is satisfied during this execution. The input domain of an element required by a structural criterion is constrained by a constraint C and the constrained element is derived. To derive a constrained element, that is, to select C, some aspects must be considered, discussed in the examples that follow.

3.2. *An Example*

Selecting a constraint C is a very important task. The efficacy of test data can depend on it and some aspects should be considered to derive C. The first one is the efficacy of test data; C should describe faults revealed by functional and fault-based techniques generally not revealed by structural criteria. In this way C could describe faults in data structures, predicates, etc. Table 1 shows some constraints that can be used to derive constrained elements. The column "Related Technique" shows some techniques that can be used with structural criteria. They are:

1. The fault-based technique can be applied using any mutation operator. In Table 1 the operator used is *variable mutation*. This operator for program *max* changes occurrences of the variables m, n, or *max*; the necessity conditions to reveal these faults are presented in column "Level 1" of Table 2. They are described in the third column in Table 1.
2. Predicate Based Testing (BOR/BRO) [15] can also be applied. BOR/BRO constraints are also described in the third column of Table 1. They are constraints describing faults in predicates of the program. In program max the predicate to be considered is $(n >= m)$ and the constraints derived are presented in column "Level 3" of Table 2.
3. Boundary Value Analysis, Domain Testing, and Computation Testing are techniques that can also be used. Other levels and techniques are described in [18].

Note that only necessity conditions are derived. These conditions, added to the conditions of the paths to cover the structural elements, have a high probability of meeting the sufficiency conditions.

Table 1. Constraint levels.

Level/ Information	Related technique	Necessary condition
1	Variable	$X = 0, X < 0, X > 0$
(i, j, k, x)	mutation	$A[e_1] > 0, A[e_1] < 0, A[e_1] = 0$
		$S.c = 0, S.c > 0, S.c < 0$
		$p = null$
2	Variable	$A[e_1] \neq B[e_1]$
a set as:	mutation	$p \neq q$
(i, j, x_1)		$A[e_1] \neq X$
(i, j, x_2)		$S.c \neq X$
...		$(*p) \neq X$
(i, j, x_n)		$Y \neq X$
		$S \neq T$
		$S.c \neq T.c$
		$S.c_1 \neq S.c_2$
3	BRO/BRE	
$(i, (j, k), x)$	C_1 or C_2	$F(C) = S_{1f}\%S_{2f}$
and		$T(C) = \{S_{2t} * \{f_2\}\}\$\{\{f_1\} * S_{2t}\}$
predicate		$f_1 \in S_{1f}, f_2 \in S_{2f},$
		$(f_1, f_2) \in F(C)$
in (j, k)	C_1 and C_2	$T(C) = S_{1t}\%S_{2t}$
		$F(C) = \{S_{1f} * \{t_2\}\}\$\{\{t_1\} * S_{2f}\}$
		$t_1 \in S_{1t}, t_2 \in S_{2t},$
		$(t_1, t_2) \in T(C)$

Table 2. Constrained-potential-associations for program *max* from Fig. 1.

Association	Level 1	Level 2	Level 3
(1, (1, 2), *n*)	$n > 0$	$n \neq m$	$n > m$
	$n < 0$	$n \neq max$	$n = m$
	$n = 0$	$max \neq m$	$n = m$
(1, (1, 2), *m*)	$m > 0$	$n \neq m$	$n > m$
	$m < 0$	$n \neq max$	$n = m$
	$m = 0$	$max \neq m$	
(1, (1, 2), *max*)	$max > 0$	$n \neq m$	$n > m$
	$max < 0$	$n \neq max$	$n = m$
	$max = 0$	$max \neq m$	
(1, (1, 3), *n*)	$n > 0$	$n \neq m$	$n < m$
	$n < 0$	$n \neq max$	
	$n = 0$	$max \neq m$	
(1, (1, 3), *m*)	$m > 0$	$n \neq m$	$n < m$
	$m < 0$	$n \neq max$	
	$m = 0$	$max \neq m$	
(1, (1, 3), *max*)	$max > 0$	$n \neq m$	$n < m$
	$max < 0$	$n \neq max$	
	$max = 0$	$max \neq m$	
(2, (-, -), *max*)	$max > 0$		
	$max < 0$		
	$max = 0$		

In addition to the efficacy, other aspects that must be considered in deriving the constraints are: information about the program being tested, information about the program domain, information needed to derive the constraint, unfeasibility of paths, etc. Unfeasibility is connected with the conditions of the paths covering E. The path condition can be inconsistent with the constraint C and an infeasible constrained element is derived. Determining if an element is infeasible is an undecidable problem; hence, reducing the number of infeasible paths, as well as the number of test cases required to satisfy the Constraint Based Criteria, is strongly desirable. In this sense, Vergilio et al. [18] suggest the application of the Constraint Based Criteria using different levels of constraints to reduce costs and the number of infeasible elements. The idea is to choose the constraint levels according to the reliability desired for the program being tested.

The complexity of a constraint based criterion depends on the set of constraints used and on the complexity of the correspondent structural criterion. For example, the complexity of data flow based criteria is exponential with respect to the number of decision statements in the program [9]; however, the empirical complexity of data flow based criteria observed during experiments is linear [10, 23, 24]. The complexity of the all-constrained-potential-uses criterion (C-PU) is also exponential; some evidences of its empirical complexity are given in the next section.

4. Applying Constraint Based Criteria

This section presents results from an experiment using the all-potential-uses criterion and the all-constrained-potential-uses criterion. The goals are to assess the applicability of constraint based criteria and to compare the efficacy of both criteria. The constraint based criteria applicability was shown in a previous experiment described in [22]. The experiment used simple programs and the generation of the elements required by the all-constrained-potential-uses criterion as well as the adequacy analysis were done manually. However, the programs used in this work are more complex. They were first used by Wong et al. [26]. An extension to POKE-TOOL [18] was implemented to support the Constraint Based Criteria. An instrumentation model for the CBC was established and an evaluation module, based on this model, was developed. A module was implemented to provide a measure of test set adequacy with respect to the all-constrained-potential-uses criterion. For greater detail see [18].

4.1. Description of the Experiment

The experiment consisted of two main phases: satisfaction of both criteria and efficacy analysis. For each program, the following steps were performed:

4.1.1. Satisfying the Criteria.

1. Generation of required elements: using POKE-TOOL, the elements required by both criteria were generated. BRO constraints [15] only were used to derive the constrained elements, as shown in Table 1 (Level 3). These constraints describe faults in composite predicates and were chosen due to their implementation simplicity. We selected the potential-uses criterion because they are "strong" criteria, include most of data flow based criteria and, consequently, has a high probability of revealing faults. In this way, to increase the efficacy of this criterion is assumed to be a more difficult task than that of increasing the efficacy of the all-edges criterion.
2. Generation of test data set: we first generated data test sets with the goal of covering the elements required by the all-potential-uses criterion (PU). The test data sets were manually generated, without applying any test data generation technique. In this step we also determined infeasible associations and obtained the PU coverage analysis.
3. Analysis of C-PU coverage: the test data sets generated in Step 2 were used to obtain an initial coverage for the all-constrained-potential-uses criterion (C-PU) and a list of remaining constrained associations. Additional test cases were generated and infeasible constrained-associations were determined.

Table 3 presents the coverage obtained for both criteria for each program.

Table 3. Results from PU and C-PU criterion applications.

Program criterion	Req. elm.	Infeas. elem.	Coverage (%)	Nr. test cases
Cal				
PU	242	69	71,49	12
C-PU	488	180	63,11	17
Checkeq				
PU	582	114	80,41	76
C-PU	1539	471	69,40	164
Comm				
PU	427	150	64,87	68
C-PU	884	383	56,67	74
Crypt				
PU	322	65	79,81	14
C-PU	650	195	70,00	19
Look				
PU	524	87	83,00	34
C-PU	1045	247	76,36	40
Spline				
PU	999	206	79,38	57
C-PU	2352	956	59,35	63
Tr				
PU	1527	933	38,90	30
C-PU	3040	2014	33,75	35
Uniq				
PU	262	32	87,79	36
C-PU	552	136	75,36	42
PU total	4885	1656	66.10	327
C-PU total	10550	45823	56.56	454

4.1.2. Analysis of Efficacy. Wong [26] has derived for each program a set of incorrect versions. Each version has only one fault corresponding to a simple mutation. Using these versions we did the efficacy analysis phase.

1. Classification of the faults in each version: the faults were classified as domain faults, computational faults [25], or data structure faults. Table 4 summarizes some characteristics of the programs: total number of sub-programs, number of code lines, number of faults or incorrect versions, number of nodes in the control flow graph, and number of predicates.
2. Execution of incorrect versions: PU and C-PU adequate test sets were used to execute the incorrect

Table 4. Main characteristics of the used programs.

Progr.	Nr. sub. prog.	Nr. lines	Nr. faults	Nr. nodes	Nr. pred.
Cal	4	203	20	59	20
Checkeq	2	104	22	61	24
Comm	6	170	19	84	24
Crypt	3	132	18	52	18
Look	4	146	20	71	17
Spline	7	332	20	119	40
Tr	3	141	19	97	32
Uniq	6	140	19	67	21
Total	35	1368	157	610	196

Table 5. Efficacy of PU and C-PU criteria.

Prg Crit.	Comp. fault	Dom. fault	D. Strc. fault	Total
Cal				
PU	12	6/7	1	19/20
C-PU	12	7	1	20
Checkeq				
PU	12/13	8/9		20/22
C-PU	12/13	8/9		20/22
Comm				
PU	8	11		19
C-PU	8	11		19
Crypt				
PU	12/13	3/4	0/1	15/18
C-PU	12/13	4	0/1	16/18
Look				
PU	10	9	1	20
C-PU	10	9	1	20
Spline				
PU	11	5/9		16/20
C-PU	11	5/9		16/20
Tr				
PU	13	6		19
C-PU	13	6		19
Uniq				
PU	9	9	0/1	18/19
C-PU	9	9	0/1	18/19
Rev. total PU	87	57	2	146
Rev. % PU	97.75%	89%	50%	92%
Rev. total C-PU	87	59	2	148 %
Rev. % C-PU	97.75%	92%	50%	94.27%
Total	89	64	4	157

versions of each program. Table 5 presents the efficacy for both test sets, the number of revealed faults and the percentage in each fault category.

4.2. *Analyzing the Results*

Table 3 shows the applicability of the all-constrained-potential-uses criterion. The number of test cases does not grow proportionally to the number of required associations. The C-PU criterion requires 115.96% more elements than the PU criterion; however, only 38.83% more test cases are required. The coverage obtained using C-PU is lower than the coverage obtained using PU, due to the existence of a larger number of infeasible constrained associations.

The generation of test cases to satisfy both PU and C-PU was done by the same tester. Once he generated test cases to satisfy the PU criterion and identified the feasible associations, little additional effort was necessary to generate test cases for C-PU and to identify infeasible constrained associations. The reason is that the tester used knowledge acquired about the program from the generation of PU test cases in the generation of C-PU test cases. Another question related to the applicability of C-PU is the time spent for the adequacy analysis. On the average, triple the time was spent with the C-PU criterion.

The C-PU criterion efficacy was generally greater than the PU criterion efficacy. C-PU revealed 2.27% more faults than PU, 3.13% more when we consider only domain faults, and no additional faults when we consider data structure faults. This fact shows the influence of the constraints used to derive the constrained elements. BOR constraints describe faults in predicates, which are associated to domain faults. The probability of revealing this kind of fault was greater for C-PU. However, constraints describing data structure faults were not used and there was no difference between the PU and C-PU criteria when we consider this kind of fault. The information given in Table 6 shows an incorrect version of a statement in *cal*, program used in the experiment. In this case, using the C-PU criterion with BRO constraints is more advantageous than

Table 6. Information about program *cal*.

Incorrect version	if $(y > 9999)$
Correct version	if $(y < 1 \|\|\|\| y > 9999)$
Constrained element	$(1, (14, 15), \{argc\}, (y < 1 \|\|\|\| y > 9999))$

using just the PU criterion. The constrained element shown, derived using BOR constraints, guarantees the fault detection.

5. Conclusion

We introduced the Constrained Based Criteria. These testing criteria explore positive aspects from different strategies and associate constraints to the elements required by a structural criterion; such constraints describe necessary conditions for revealing typical faults. A constrained element (E, C), required by a constraint based criterion, is covered by a test case that executes a path exercising the structural element E and satisfies the constraint C. The preliminary results concerning the applicability of the Constraint Based Criteria are very encouraging. The number of test cases required does not increase proportionally to the number of required elements. The C-PU criterion revealed 3% more domain faults than the PU criterion. This percentage is very significant, if we are testing systems which demand a high reliability level or if we do not have enough time to apply all the structural criteria; for instance, we can choose a weak criterion such as the all-edges criterion and improve its efficacy by using the all-constrained-edges criterion.

The task of selecting constraints is very important because they will influence the efficacy of the constrained criterion. Constraints must be chosen to describe faults complementary to those revealed by structural testing; for example, faults in data structures. In addition to this, other factors must be considered: common faults found in similar programs to the program being tested, number of infeasible elements, or selective constraints (selective Constraint Based Criteria). Applying selective Constraint Based Criteria means that specific constraints only are used. This may reduce costs, by decreasing the number of required elements and the number of test cases. Other comparisons are being carried out using a random test data generation strategy. The same test data sets and programs used in the experiment described in Section 4 will be submitted to the Proteum tool for comparing the Constraint Based Criteria and the Mutation Analysis Criteria.

The combination of necessity conditions and path conditions usually meets sufficiency conditions. This is the main advantage of using the Constraint Based Criteria, i.e., the increase in the efficacy of structural testing by using test cases with a high probability of revealing faults. The Constraint Based Criteria presented in this paper provides a way to find faults other than the faults revealed by structural testing by using the distinct principles of the different test data generation strategies. They can be used either to select test data sets with a high probability of revealing faults or to assess test data sets adequacy.

References

1. M.L. Chaim, "POKE-TOOL—Uma Ferramenta para Suporte ao Teste Estrutural de Programas Baseado em Análise de Fluxo de Dados," Master Thesis, DCA/FEEC/Unicamp, Campinas—SP, Brazil, April 1991 (in Portuguese).
2. M.E. Delamaro and J.C. Maldonado, "A Tool for the Assesment of Test Adequacy for C Programs," *Proceedings of the Conference on Performability in Computing Systems*, East Brunswick, NJ, USA, July 1996, pp. 79–95.
3. R.A. De Millo and A.J. Offutt, "Constraint-based Automatic Test Data Generation," *IEEE Transactions on Software Engineering*, vol. SE-17(9), pp. 900–910, September 1991.
4. R.A. De Millo and A.J. Offutt, "Experimental Results on Automatic Test Case Generation," *ACM Transactions on Software Engineering and Methodology*, vol. SE-2(2), pp. 109–127, April 1993.
5. F.G. Frankl and E.J. Weyuker, "Data Flow Testing in the Presence of Unexecutable Paths." *Proceedings of the Workshop on Software Testing*, July 1986, Banff, Canada Computer Science Press: pp. 4–13.
6. P.G. Frankl and E.J. Weyuker, "A Formal Analysis of the Fault-Detecting Ability of Testing Methods," *IEEE Transactions on Software Engineering*, vol. SE-19(3), pp. 202–213, March 1993.
7. J.R. Horgan and S. London, *ATAC—Automatic Test Coverage Analysis for C Programs*, June 1990.
8. J.W. Laski and B. Korel, "A Data Flow Oriented Program Testing Strategy," *IEEE Transactions on Software Engineering*, vol. SE-9(3), pp. 347–354, May 1983.
9. J.C. Maldonado, "Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software," Doctorate Dissertation, DCA/FEEC/Unicamp, Campinas—SP, Brazil, July 1991 (in Portuguese).
10. J.C. Maldonado, M.L. Chaim, and M. Jino, "Briding the Gap in the Presence of Infeasible Paths: Potential Uses Testing Criteria," *XII International Conference of the Chilean Science Computer Society*, Santiago, Chile, October 1992, pp. 323–340.
11. S.C. Ntafos, "On Required Element Testing," *IEEE Transactions on Software Engineering*, vol. SE-10(6), pp. 795–803, November 1984.
12. R.B. Pressman, *Software Engineering: A Practitioner's Approach*, 3rd Ed., New-York: McGraw-Hill, EUA, 1992.
13. S. Rapps and E.J. Weyuker, "Selecting Software Test Data using Data Flow Information," *IEEE Transactions on Software Engineering*, vol. SE-11(4), pp. 367–375, April 1985.
14. S.R.S. Souza, J.C. Maldonado, and S.R. Vergilio, "Análise de Mutantes e Potenciais-usos: Uma Avaliação Empírica," *VIII Cconferência Internacional de Tecnologia de Software: Qualidade de Software*, Curitiba—PR, June 1997 (in Portuguese).

15. K.C. Tai, "Predicate-Based Test Generation for Computer Programs," *Proceedings of International Conference on Software Engineering*, May 1993, IEEE Press, pp. 267–276.
16. K.C. Tai, "Theory of Fault-Based Predicate Testing for Computer Programs," *IEEE Transactions on Software Engineering*, vol. 22(8), pp. 552–562, August 1996.
17. H. Ural and B. Yang, "A Structural Test Selection Criterion," *Information Processing Letters*, vol. 28(3), pp. 157–163, July 1988.
18. S.R. Vergilio, "Critérios Restritos de Teste de Software: Uma Contribuição para Gerar Dados de Teste mais Eficazes," Doctorate Dissertation, DCA/FEEC/Unicamp, Campinas—SP, Brazil, July 1997 (in Portuguese).
19. S.R. Vergilio, J.C. Maldonado, and M. Jino, "Resultados da Aplicação de Diferentes Técnicas de Geração de Dados de Teste Sensíveis a Erros," Technical Report, DCA/FEEC/Unicamp, Campinas—SP, Brazil, 1996 (in Portuguese).
20. S.R. Vergilio, J.C. Maldonado, and M. Jino, "An Experimental Evaluation of Different Testing Techniques," Technical Report, DCA/FEEC/Unicamp, Campinas—SP, Brazil, 1996.
21. S.R. Vergilio, J.C. Maldonado, and M. Jino, "Infeasible Paths Within the Context of Data Flow Based Criteria," *VI International Conference on Software Quality*, October 1996, Ottawa-Canada, pp. 310–321.
22. S.R. Vergilio, J.C. Maldonado, and M. Jino, "Constraint Based Selection of Test Sets to Satisfy Structural Software Testing Criteria," *XVII International Conference of the Chilean Computer Science Society*, Valparaiso, Chile, November 1997, Los Alamitos, CA: IEEE-Press.
23. E.J. Weyuker, "An Empirical Study of the Complexity of Data Flow Testing," *Proceedings of the Second Workshop on Software Testing, Verification and Analysis*, July 1988, Banff, Canada: Computer Science Press, pp. 188–195.
24. E.J. Weyuker, "The Cost of Data Flow Testing: An Empirical Study," *IEEE Transactions on Software Engineering*, vol. SE-16(2), pp. 121–128, February 1990.
25. L.J. White and E.I. Cohen, "A Domain Strategy for Computer Program Testing," *IEEE Transactions on Software Engineering*, vol. SE-6(3), pp. 247–257, May 1980.
26. W.E. Wong, A.P. Mathur, and J.C. Maldonado, "Mutation Versus All-Uses: An Empirical Evaluation of Cost, Strength and Effectiveness," *Software Quality and Productivity—Theory, Practice, Education and Training*, Hong Kong, December 1994.

Silvia Regina Vergilio received the the MS (1991) and DS (1997) degrees from University of Campinas, UNICAMP, Brazil. She is currently at the Computer Science Department at the Federal University of Paraná, where she has been a faculty member since 1993. She has been involved in several projects and her research interests are in the areas of Genetic Programming and Software Engineering as software testing, software quality and software metrics.

José Carlos Maldonado received his BS in Electrical Engineering /Electronics in 1978 from the University of São Paulo (USP), Brazil, his MS in Telecommunications/Digital Systems in 1983 from the National Space Research Institute (INPE), Brazil, and his D.S. degree in Electrical Engineering/Automation and Control in 1991 from the University of Campinas (UNICAMP), Brazil. He worked at INPE from 1979 up to 1985 as a researcher. In 1985 he joined the Computer Science Department of the Institute of Mathematical Sciences and Computing at the University of São Paulo (ICMC-USP) where he is currently a faculty member. His research interests are in the areas of Software Quality and Software Testing and Reliability. He is a member of the SBC (Brazilian Computer Society) and of the IEEE Computer Society.

Mario Jino received the BS degree in electronic engineering from Instituto Tecnológico de Aeronáutica (ITA), Brazil, in 1967, the MSc in electrical engineering from the State University of Campinas (UNICAMP), Brazil, in 1974, and the PhD degree in computer sciences from the University of Illinois, Urbana-Champaign, Illinois, USA, in 1978. From 1967 to 1971 he was a researcher at the Brazilian Space Agency (INPE), where he was involved in the Remote Sensing Project. Since 1971 he is a teacher at UNICAMP where he is presently an associate professor in the Department of Computer Engineering and Industrial Automation of the School of Electrical and Computer Engineering. His current research interests include: software quality; software testing, debugging and maintenance; object orientation; and software metrics. He has been involved in several technological development projects with government and private entities, and has served as technical advisor/referee for Brazilian research funding agencies as well as in several scientific conferences and symposiums. He is a member of the IEEE Computer Society, the IEEE, the ACM, SBA (the Brazilian Society of Automatic Control), and SBC (the Brazilian Computer Society).