JOURNAL OF ELECTRONIC TESTING: Theory and Applications 20, 439–445, 2004

Mutation Analysis and Constraint-Based Criteria: Results from an Empirical Evaluation in the Context of Software Testing

INALI WISNIEWSKI SOARES
UNICENTRO, Guarapuava CP: 3010, 85010-990, Brazil
inali@unicentro.br

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

Received September 9, 2002; Revised May 12, 2003

Editors: F. Vargas and V. Champac

Abstract. Several software testing criteria have been proposed during last years with the goal of aiming the test set generation and revealing many faults as possible. They are considered complementary because can reveal different kind of faults and are based on different principles. For example, structural criteria use the internal structure of the program for deriving test cases; Mutation Analysis is a fault-based criterion; and Constraint Based Criteria use constraints to be satisfied during the program execution. Because of this, some questions can be posed, such as: "What criterion should be used or be first applied?". Many research works compare criteria with the goal of answering these questions. However, some criteria as Mutation Analysis and Constraint Based Criteria are theoretically incomparable and only empirical studies can point out the relation between them. This work presents results from an empirical evaluation of Mutation Analysis and All-Constrained- Potential-Uses criterion considering the factors: cost (number of test cases), efficacy (number of revealed faults) and strength (difficulty of satisfying a criterion, given that another one has been satisfied). The obtained results show an empirical relation, which is used to propose a strategy for application of different testing criteria.

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

1. Introduction

The use of software products in most areas of human activities has generated a growing interest in software quality assurance. In this context, the software testing activity is fundamental and has gained importance during the last decade. The main goal of testing is to find an unrevealed fault [6] and consequently to increase the software reliability.

Testing techniques were proposed with this goal, that is, to reveal a great number of faults with minimal effort and costs. There are three groups of techniques: functional technique: uses functional specification by aspects of a program to derive test cases; structural technique: derives test cases based on paths in the control-flow graph of the program; 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 that are predicates to consider the testing activity ended, that is, to consider a program tested enough and to help the tester in the task of test case selection.

Structural criteria require the execution of paths in the program. These paths must exercise elements of the source code or the control-flow graph of the program being tested. They are control-flow and data-flow based criteria. All- Uses and All-Potential-Uses are examples of data-flow based criteria [4, 7].

Mutation Analysis is a fault-based criterion [2]. It establishes mutant programs that 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 producing a mutant behavior that is different from the original program behavior.

With the goal of increasing the efficacy of structural testing criteria, Vergilio et al. introduced the Constraint-Based Criteria [9]. They associate a constraint to an element required by a structural criterion. This constraint must be satisfied during the testing and can describe different kind of faults.

The criteria are considered complementary because they can reveal different kind of faults. Due this, many questions can be posed during the testing activity. Examples of these questions are: "What criteria should be used?" and "What criterion should be first applied?". The answer is not simple and depends on different factors. In this sense, comparisons between testing criteria are fundamental [5]. They are usually compared using three factors [10]: (1) cost: related to the number of test cases required to satisfy the criterion; (2) efficacy: the ability to reveal faults; (3) strength: related to the difficulty of satisfying a criterion, given that another one has been satisfied. This last factor is related to the inclusion relation among criteria.

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 inclusion relation among testing criteria are described in [4, 7].

There are in the literature many theoretical and empirical works comparing testing criteria [4, 5, 9, 10]. The data-flow based criteria are stronger than control-flow based criteria, that is, they include control-flow based criteria. A constraint-based criterion includes the correspondent structural criterion [9]. However, Mutation Analysis is incomparable with Constraint-Based Criteria and with structural criteria.

Vergilio et al. [9], accomplished an experiment comparing All-Constrained Potential Uses with All-Potential-Uses, its correspondent structural criterion. In this study, the constrained criterion shows a greater efficacy and cost than the structural criterion. Other empirical studies show that Mutation Analysis is more expensive and efficacious than structural criteria [10]. However, there is no work comparing Mutation Analysis with Constraint-Based Criteria. They are incomparable but, empirical studies can show the relationship between them.

This work presents the results from an empirical evaluation of Mutation Analysis and the All-Constrained Potential-Uses criterion, considering the factors cost, efficacy and strength. The same programs and test cases, from the experiment of Vergilio et al. [9] were used, and this also made possible the comparison between Mutation Analysis and All-Potential-Uses criterion. We obtained an empirical relation among the compared criteria. Based on this result, we introduce a testing strategy for the criteria application with the goal of answering the above questions.

The paper is organized as follows. Section 2 shows the main concepts and terminology about software testing criteria. Section 3 describes the accomplished experiment. Section 4 presents the obtained results. Section 5 concludes the paper, by introducing a testing strategy based on the results of Section 4.

2. Testing Criteria

Testing criteria have the goal of selecting "good test cases." They are predicates to be satisfied. The criteria are usually derived applying different testing techniques: functional, structural and fault-based. They require elements to be tested and offer coverage measures.

The structural technique uses the source code and a program graph (G) to derive test cases. A criterion can be used to generate test data or to evaluate the adequacy of a test set. A test set T is called C-adequate test set, if T satisfies C. Control-flow based criteria and data-flow based criteria are the best known structural criteria. Control-flow based criteria require the exercising of elements in the control-flow graph. Data-flow based criteria test associations between definitions and uses of variables in a program.

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: requires the execution of all paths of G.

Data-Flow Based Criteria

- All-Uses: requires the execution of a path in G that covers 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 [7].
- All-Potential-Uses: requires the execution of a path in G that covers 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 [4].

Structural testing criteria generally require exercising elements of the source code of the program being tested. A path of the program must cover these elements. Constraint- Based Criteria [9] were defined by associating a constraint to the element required by a structural testing criterion. They require pairs (E, C) (element, constraint) which are covered by a test case that executes a path covering the element E and satisfying C.

The constraint C can describe any kind of fault. More than one constraint can be associated to an element. In this way, test sets generated to satisfy the structural criterion have high probability of revealing the fault described by C and consequently the efficacy of the structural criterion will be greater. For example, consider the predicate $(a \geq 5)$ that must be evaluated true to cover a required element E. Two pairs will be additionally required by the constrained criteria $(E, (a > 5))$ and $(E, (a == 5)))$. The pairs reveal a fault such as the replacement of the relational operator $>$ by $\geq$. Tai [8] presents two techniques to reveal faults in predicates, such as the shown above. They are: BOR (Boolean Operator Testing) and BRO (Boolean and Relational Operator Testing). Constraint-Based Criteria can be derived from any structural criteria with the goal of increasing its efficacy. Constraint-Based Criteria for the best known structural criteria are presented next:

Constrained Control-Flow Based Criteria. All-Constrained-Nodes and All-Constrained-Edges criteria are defined. For these criteria constrained elements are, respectively constrained nodes, 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.

Several tools have been developed to support data-flow based criteria. We use Poketool [1], a multi-language tool, that supports the control-flow based criteria: All-Edges, All-Nodes, and the “Potential Use Criteria” [4]: All-Potential-Uses, All-Potential-Uses/du, All-Potential-du-Paths. This tool was extended to permit the adequacy analysis with respect to the Constrained Potential Uses Criteria.

Using Poketool, several experiments were conducted to validate the applicability of the structural and of the Constraint Based Criteria [4, 9]. These experiments compare the All-Constrained-Potential-Uses with All-Potential-Uses. The programs, data and results of these experiments were used in our work and are presented in next section.

Fault-based criteria require the program execution to show the presence or absence of a fault in the program P, being tested. Mutation Analysis is a fault-based criterion. It establishes mutant programs that are versions of the original program P. They differ from P by just a simple modification [2]. 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 the original program behavior. Proteum [3] is a tool that implements Mutation Analysis. It has several mutation operators and permits the generation and execution of mutants. After the execution, a mutation score can be obtained, given by the number of dead and alive mutants.

Determining infeasible paths and equivalent mutants are undecidable questions. This fact causes many problems to the testing criteria automation, mainly for generation of test cases to satisfy a criterion. These aspects are limitations for the criteria application and consequently for the use of the tools mentioned above. The determination of infeasible elements and equivalent mutants, as well as the generation of adequate test sets are usually done by the tester.

3. Description of the Experiment

The experiment allowed the comparison between the All-Constrained-Potential-Uses criterion (CPU criterion) and Mutation Analysis (MA) considering the factors cost, efficacy, and strength. Comparing the Mutation Analysis with All-Potential-Uses (PU) criterion was also possible.

Table 1 presents some characteristics of the used programs: a classification for the faults introduced, their sizes given by the number of LOC (Lines of Code) and the complexity of McCabe. They are 8 UNIX programs first used by Wong [10] and after by Vergilio et al. [9]. Due to this, we used in our analysis the same PU and CPU adequate test sets and results obtained by Vergilio et al. PU and CPU applications were not necessary. Vergilio et al. used Poketool and generate the data test sets manually.

Our experiment consisted basically in the MA application using Proteum and the same set of programs and test sets. The experiment was conducted in four steps:

Table 1. Size of the programs and introduced faults.

Prg	LOC	McCabe complex.	Comp. fault	Dom. fault	D. Strc. fault	Total
cal	203	28	17	11	2	20
checkeq	104	26	14	17	1	22
col	301	65	28	17	2	37
comm	170	40	14	15		19
look	146	33	13	16	1	20
spline	332	64	18	11	1	20
tr	141	44	17	11	1	19
uniq	140	34	11	16	2	19
Total	1537	334	132 (51.56%)	114 (44.53%)	10 (3.91%)	256

(1) generation of the mutants: the mutants were generated using all operators available in Proteum.

(2) generation of adequate test case sets for Mutation Analysis (MA-adequate test set): initially the same sets generated by Vergilio were used to verify the strength of PU and CPU criteria with respect to Mutation Analysis. The mutation scores for PU and CPU adequate test sets were obtained. Additional test cases were generated to kill the remaining alive mutants. The equivalent mutants were also determined in this step. This activity was done by hand and consumed a lot of effort and time.

Table 2 shows the results obtained from the criteria application. The results from PU and CPU are extracted from [9]. For example, for program *cal*, the PU criterion requires 242 associations, 69 are infeasible, 12 test cases are necessary, and a score of 71.49% is obtained.

Table 2. Results from PU, CPU and MA criterion applications.

Program	Criterion	Req. elem.	Infeas. elem.	Coverage (%)	Nr. test cases
cal	PU	242	69	71.49	12
	CPU	488	180	63.11	17
	MA	4324	309	93.00	32
checkeq	PU	582	114	80.41	76
	CPU	1539	471	69.40	164
	MA	3075	214	93.00	74
col	PU	999	125	87.49	69
	CPU	1807	402	77.75	75
	MA	6910	887	87.00	78
comm	PU	427	150	64.87	68
	CPU	884	383	56.67	74
	MA	1938	281	85.00	47
look	PU	524	87	83.00	34
	CPU	1045	247	76.36	40
	MA	2030	216	89.00	43
spline	PU	999	206	79.38	57
	CPU	2352	956	59.35	62
	MA	12560	1426	89.00	81
tr	PU	1527	933	38.90	30
	CPU	3040	2014	33.75	35
	MA	4422	790	82.13	72
uniq	PU	262	32	87.79	36
	CPU	552	136	75.36	42
	MA	1623	160	90.00	36
Total PU		5562	1716	69.15	382
Total CPU		11707	4789	59.09	509
Total MA		36882	4283	88.39	463

Table 3. Mutation scores for the PU and CPU adequate test sets.

	Mutation score for adequate test sets		
Program	PU set	CPU set	MA set
cal	90.0	92.0	93.0
checkeq	88.0	90.0	93.0
col	79.0	81.0	85.0
comm	82.0	82.0	87.0
look	83.0	83.0	89.0
spline	85.0	85.0	89.0
tr	64.0	71.0	82.0
uniq	88.0	89.0	90.0
mean	82.3	84.1	88.5

The CPU criterion requires 488 constrained associations, 180 are infeasible, 17 test cases are necessary and a 63.11% coverage is obtained. The MA criterion requires 4324 mutants, 32 test cases are necessary, 309 mutants are equivalent and a mutation score of 93% is obtained. Table 3 shows the mutation score obtained using PU and CPU adequate test sets. The last column presents the MA score. This score is not 100%, because we did not discount the number of equivalent mutants (the same happens on Table 2).

(3) efficacy analysis: Wong [10] derived to each program a set of incorrect versions. Each version has only one fault that corresponds to a simple modification. Some undergraduate students randomly generated the other versions. Using all the versions we did the efficacy analysis.

The faults were classified in domain faults, computational faults and faults on data structures. Table 1 presents a summary of the faults and category for the programs. After this, the PU, CPU and MA adequate test sets were used to execute the incorrect versions for each program. Table 4 presents the efficacy results: number and percentage of revealed faults in each category by criterion. For example, program *cal* has 17 computational faults, all of them revealed by the PU-adequate test set. It also has 11 domain faults, however 9 was revealed by PU. It has a total of 30 faults and PU revealed 28.

4. Analysis of the Results

Next, we analyze the results presented in last section with respect to cost, efficacy and strength.

Table 4. Efficacy of PU, CPU and MA criteria.

Prg	Crit.	Comp. fault	Dom. fault	D.Strc. fault	Total
cal	PU	17	9/11	2	28/30
	CPU	17	11	2	30
	MA	17	11	2	30
checkeq	PU	13/14	13/17	0/1	26/32
	CPU	13/14	13/17	0/1	26/32
	MA	14	15/17	1	30/32
col	PU	28	17	2	47
	CPU	28	17	2	47
	MA	28	17	2	47
comm	PU	12/14	15		27/29
	CPU	12/14	15		27/29
	MA	13/14	15		28/29
look	PU	13	16	1	30
	CPU	13	16	1	30
	MA	13	16	1	30
spline	PU	18	8/11	1	27/30
	CPU	18	8/11	1	27/30
	MA	18	11	1	30
tr	PU	16/17	11/12		27/29
	CPU	16/17	11/12		27/29
	MA	17	12		29
uniq	PU	11	14/15	0/2	25/29
	CPU	11	14/15	0/2	25/29
	MA	11	15	1/2	27/29
rev. total PU		128/131	103/116	6/9	237/256
rev. % PU		97.71%	88.79%	66.67%	92.58%
rev. total CPU		128/131	105/116	6/9	239/256
rev. % CPU		97.71%	92%	66.67%	93.36%
rev. % MA		131	112/116	8/9	251/256
rev. % MA		100%	96.55%	88.89%	98.05%

(1) *Cost*: Table 5, obtained from Table 2 shows a summary of the results with respect to the criteria costs. The second column presents the mean number of test cases for the criteria considering all programs in the experiment. The third one presents the mean number excluding the program *checkeq*.

When we consider all the programs, the MA cost is 21.20% greater than PU cost and 9.94% lower than CPU cost. This fact is due to the program *checkeq* that has a great number of nested if commands. This structure maximizes the number of elements required by the PU criterion and consequently by CPU criterion. This

Table 5. PU, CPU and MA costs.

	Number of test cases	
Criteria	All programs	Excluding *checkeq*
PU	382	306
CPU	509	345
MA	463	389

program also has a lot of composed predicates. Vergilio et al. use BOR and BRO constraints [8] to generate the constrained associations. This kind of constraint is associated to the predicates of the program. The greater the number of composed predicates, the greater the number of constrained elements. If we exclude the *checkeq* program theMA cost is 27.12% greater than PU cost and 12.75% greater than CPU cost. In general, we observe that the MA cost is greater than CPU cost, not considering the influence of the special program *checkeq*.

(2) *Efficacy*: We can observe in Table 4 that the MA criterion revealed 3.78% more faults than CPU and 5.47% than PU. The CPU criterion revealed 0.78% more faults than PU. When we consider the kind of faults, we can notice:

- MA criterion revealed 4.55% and 7.76% more domain faults, respectively, than CPU and PU. CPU criterion revealed 3.21% more domain faults than PU.
- MA criterion revealed 2.25% more computation faults, than CPU and PU. There is no difference between CPU and PU for this kind of fault.
- MA criterion revealed 22.22% more data structure faults than CPU and PU.

We can conclude that the MA criterion efficacy is greater than CPU efficacy. CPU efficacy is greater than PU efficacy mainly for domain faults. This is in due the constraints used to generate the constrained-elements (BRO and B0R constraints, that are related to predicates and domain faults). CPU criterion can be more efficacious than PU but less expensive than MA. Other aspect to be considered is the influence of the constraints used to derive the constrained-elements on the efficacy.

(3) *Strength*: We can observe in Table 3 that the CPU mean mutation score is 1.8% greater than the PU mean mutation score. The mean mutation score obtained using the MA adequate test set is 6.2% and 4.4% greater than respectively the CPU and PU mean mutation score. This shows that the scores obtained using PU and CPU adequate test sets are very high (82.3%) for PU and (84.1%) for CPU. If we eliminate the equivalent mutants, we can note that 4.4% of mutants were not covered by CPU adequate sets and 6.2% were not covered by PU adequate sets. We observed in other experiments that, when the mutation score is around 80%, to increase it is a very hard task. So, the difference between PU and CPU scores is significant.

We can conclude for these programs and these data sets that it is easier to satisfy the MA criterion given that CPU criterion was satisfied, than given that PU criterion was satisfied. This shows an empirical relation that can be used to propose an application strategy for these criteria.

Based on the results, of this section, we consider the Constraint-Based Criteria as intermediate criteria between data-flow based criteria and Mutation Analysis in terms of the studied factors.

5. Conclusion

This work presented results from an experiment with Mutation Analysis criterion and All-Constrained-Potential-Uses and All-Potential-Uses criteria, considering the factors cost, efficacy and strength.

With respect to cost, MA is found as the most expensive criterion. If we exclude the program *checkeq* influence the MA cost is 27.12% greater than PU cost and 12.75% greater than CPU cost. With respect to efficacy, MA is the most efficacious. It revealed 3.78% and 5.47% more faults respectively than CPU and PU.With respect to strength, we can conclude that it is easier to satisfy MA given that CPU was satisfied than given that PU was satisfied. The results show an empirical relation among the criteria: the CPU criterion can be viewed as intermediate between MA and PU criteria. Based on this result, we introduce a strategy for application of the testing criteria, including the following steps.

I. Generation of an Initial Test Set T_i

This initial data can be generated randomly or by using functional technique. The second approach is more interesting because applying a functional testing can help the revealing of different faults not generally revealed by structural or fault-based testing criteria. Random generation is practical but it does not usually generate the expected output.

II. Testing Criteria Application

The application of the testing criteria must follow the hierarchy given by the inclusion relation between criteria [4] and the empirical hierarchy observed in our experiment. The order for application is given next.

1. Control-Flow Based Criteria: All-Nodes, All-Edges, etc.
2. Constrained Control-Flow Based Criteria: All-Constrained-Nodes, All-constrained-Edges, etc.
3. Data-Flow Based Criteria: All Defs, All-Uses, All- Potential-Uses, All-du-Paths, All-Potential-du-Paths, etc.
4. Constrained Data-Flow Based Criteria: All-Constrained-Uses, All-Constrained-Potential-Uses, etc.
5. Fault-Based Criteria: Mutation Analysis, etc.

The last criteria (CPU and MA) are the strongest criteria; they are more expensive and efficacious. However, applying all the criteria is not always necessary. It will depend on the desired reliability and on the characteristics of the program. To apply the criteria, the use of a tool, as Poketool or Proteum, is very important to decrease the effort and time spent in the testing activity.

To apply a criterion:

(a) generate the required elements
(b) obtain an initial coverage using T_i.
(c) generate additional test sets T_a to cover the remaining elements or mutants required by the criterion. During this step determine infeasible elements or equivalent mutants.
(d) obtain the final coverage using T, such as $T = T_i + T_a$.

To apply the next criterion in the hierarchy, use T as the new initial test set T_i in Step b and it will be easier to satisfy the next criterion.

These steps help the tester to choose the order for the testing criteria application and allows the use of the different testing techniques and principles.

Other experiments with other kind of programs should be accomplished to validate this strategy and to confirm the obtained results. A study about equivalent mutants is also being conducted. This study can provide mechanisms to be implemented in Proteum to reduce the effects of equivalent mutants in the testing activity.

Acknowledgments

We would like to thank Jose Carlos Maldonado, USP São Carlos, Brazil and W. Eric Wong, Texas University, USA, for providing the programs and some test cases used in this work.

References

1. M.L. Chaim, "POKE-TOOL—A Tool for Supporting Data Flow Based Testing," Master Thesis, DCA/FEEC/Unicamp, Campinas—SP, Brazil, April 1991 (in Portuguese).
2. R.A. De Millo, R.J. Lipton, and F.G. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer," *IEEE Computer*, vol. C-11, pp. 34–41, 1978.
3. M.E. Delamaro and J.C. Maldonado, "A Tool for the Assesment fo Test Adequacy for C Programs," in *Proceedings of the Conference on Performability in Computing Systems*, East Brunswick, New Jersey, USA, July 1996, pp. 79–95.
4. J.C. Maldonado, M.L. Chaim, and M. Jino, "Briding the Gap in the Presence of Infeasible Paths: Potential Uses Testing Criteria," in *XII International Conference of the Chilean Science Computer Society*, Chile: Santiago, IEEE Press, Oct. 1992, pp. 323–340.
5. A.P Mathur and W.E. Wong, "An Empirical Comparison of Data Flow and Mutation Based Test Adequacy Criteria," *The Journal of Software Testing, Verification and Reliability*, vol. 4, no. 1, pp. 9–31, 1994.
6. G.J. Myers, *The Art of Software Testing*, Wiley, 1979.
7. S. Rapps and E.J. Weyuker, "Selecting Software Test Data Using Data Flow Information," *IEEE Trans. on Software Engineering*, vol. SE-11, no. 4, pp. 367–375, 1985.
8. K.C. Tai, "Predicate-Based Test Generation for Computer Programs," in *Proceedings of International Conference on Software Engineering*, IEEE Press, May 1993, pp. 267–276.
9. S.R. Vergilio, J.C. Maldonado, and M. Jino, "Constraint Based Criteria: An Approach for Test Case Selection in the Structural Testing," *Journal of Eletronic Testing*, vol. 17, no. 2, pp. 175–183, 2001.
10. W.E. Wong, A.P. Mathur, and J.C. Maldonado, "Mutation Versus All-Uses: An Empirical Evaluation of Cost, Strength and Effectiveness," in *Software Quality and Productivity—Theory, Practice, Education and Training*, Hong Kong, Dec. 1994.