Estimating Effort of Incremental Integration Software Testing and Design Metrics

ZORICA M. MIHAJLOVIĆ[1], DUŠAN M. VELAŠEVIĆ[2] and NIKOS E. MASTORAKIS[3]
[1]Computer Systems Design Lab.
VINČA Institute of nuclear sciences
P.O. Box 11001, Belgrade
SERBIA and MONTENEGRO (YUGOSLAVIA)

[2]Faculty of Electrical Engineering
University of Belgrade
Bulevar kralja Aleksandra 73, 11000 Belgrade
SERBIA and MONTENEGRO (YUGOSLAVIA)

[3]Department of Computer Science of the Military Institute
University of Education, Hellenic Naval Academy
Hatzikyriakou Avenue, 18539, Piraeus
GREECE

Abstract: - Estimating effort is an important component of planning software engineering tasks. Preventive incremental integration software testing is one of such tasks. In the moment when this testing begins software design already exists and can be used in effort estimating. Seven design metrics already proposed in the literature on software engineering have been selected in this paper in order to analyze their applicability to estimating effort of incremental integration testing. A number of programs has been developed to collect the data needed for this analysis. In addition, the conditions under which these data have been collected are shown. Based upon the data, the metrics have been analyzed by using the correlation between each of the metrics and the actual effort spent on this testing. The results point to best metrics to be used for the estimation purpose.

Key - Words: - Software Testing, Incremental Integration Testing, Estimating Effort, Design Metrics.

1 Introduction

Estimating effort is an important component of planning in software engineering [20], [9]. It is important when a software project is considered as well as when a separate task of the project is treated. The generic task of software testing is an example of such a task. Estimating effort helps in determining the calendar time of the task, predicting the cost of the task and refining the task's budget.

The task of software testing can further be decomposed into the levels of testing [17], [19], [24]. Unit and integration testing are two lowest levels of testing. According to experts [17], for software developed through procedural paradigm the best way to perform unit and integration testing is to consider them as an integrated level of incremental integration testing. A program is built through a process of creating the program's increments in incremental integration testing. A new increment is created in each step by adding a new unit to the previous increment. This new increment and its interfaces are in the focus of testing. After testing and correcting this increment, a new increment is created until the complete program is formed. In this creation, when we say "a unit", we mean a software element not further decomposed into the other elements.

The prevention strategy of testing as defined in [6] is proclaimed to be the most advanced testing strategy that should be practiced. Testing

is seen in parallel with software development and the beginning of testing is shifted towards the beginning of software development. In addition, testing activities are defined analogously to software development activities. They include test planning and test design as the activities that precede test execution. Finding software problems during testing is shifted towards test planning and test design instead of leaving them for test execution only. In the context of incremental integration testing the earliest moment when this testing can begin is the moment when engineers finish software design.

Estimating effort of preventive incremental integration testing is considered in this paper. Since software design already exists in the moment when this testing begins, estimating effort is related to design metrics that can be directly derived from software design. The problem of estimating effort of incremental integration testing has not been enough studied in the previous work on software testing. The exception is given in [13] and [15] where two metrics for estimating effort of incremental integration testing have been proposed: estimated number of tests and number of units. The other previous approaches are focused on:

1. unit testing ([10], [7], [20], [3] and [23], among others),
2. system testing ([16], [21], [5], [2], [22], [17], [8] and [25], among others), and
3. consider testing within a software project without distinguishing the levels of testing ([19] and [20], among others).

In order to analyze the applicability of design metrics to estimating effort of preventive incremental integration testing a number of software design metrics that has already been proposed in the literature on software engineering is selected in this paper (Section 2). The applicability is analyzed by computing the correlation between a metric and the actual effort spent on this testing. To collect the data needed for this computation a sequence of example programs has been developed. Incremental integration testing has been included in this development. A special methodology, the ITeM methodology [13], has been used during this testing. The use of the ITeM methodology determined the conditions under which the data used for the correlation computation have been collected (Section 3). The estimation capabilities of the metrics seen through the correlation are further computed (Section 4) and the most promising of them chosen (Section 5).

2 Design Metrics

Four criteria have been set in selecting the design metrics to be included in the analysis of the applicability to estimating effort of incremental integration testing:

1. metrics should be taken from the literature on software engineering (already proposed),
2. metrics should already be mentioned in the context of software testing or software modifications,
3. metrics should be directly derived from software design and
4. metrics should be appropriate for incremental integration testing.

The third criterion requires further explanations. The starting point for the design metric consideration is the standard presentation of software design [20], [4]. This presentation includes a structure chart with couples and the corresponding detailed design of units shown in the form of PDL (Program Design Language). The design metric consideration excludes any special presentations of software design. The presentation with loops and decisions within a structure chart is an example of such a special presentation. This special presentation has been introduced in [11] to extend the cyclomatic complexity metric into the design level. In addition, the third criterion about the metrics directly derivable from software design means that it is enough to go through design to capture a metric. Special computations, like the computation of cohesion of units with data flow analysis between statements of a unit [3], are excluded.

Starting from the enumerated criteria, the set of metrics for the analysis of their applicability to estimating effort of incremental integration testing has been defined:

1. the number of lines of PDL derivable from the detailed design of units – LOP (Lines of PDL),
2. cyclomatic complexity of units [10] – CC,
3. the $(f_i * f_o)^2$ metric introduced in [7] - F2,

4. the LOP * $(f_i * f_o)^2$ metric inspired by [7] - LF2,
5. the modified quantification of coupling introduced in [3] – MCp,
6. the estimated number of tests proposed for estimating effort of incremental integration testing in [13] and [15] – ENT, and
7. the number of units in a structure chart, the direct chart metric [20] already proposed for estimating effort of incremental integration testing in [13] and [15] – NU.

The LOC metric (Lines of Code) is a standard metric used in estimating effort in software engineering especially in software project effort estimation [20]. It is also used in estimating effort of smaller tasks of software engineering that are assigned to a single software engineer as is defined in Personal Software Process – PSP [9]. Independently of the task dimension, the number of LOC is estimated first. Then, based upon this number and the historical data collected from previous similar tasks, the task's effort is estimated. The LOP metric is included analogously to the LOC metric. Since detailed software design already exists in the moment of planning incremental integration testing, finding the number of LOP is easy as it can be directly derived from this design.

Cyclomatic complexity is a well-known metric that describes the complexity of units starting from the number of conditions of code [10]. It is considered in the context of unit testing. It has been shown that units with higher cyclomatic complexity are less reliable than those with lower cyclomatic complexity. The number of 10 can be taken as an approximate limit. Although cyclomatic complexity has originally been proposed as the code metric, it can be applied to detailed software design presented in PDL. Because of that, cyclomatic complexity of units is included in the set of metrics potentially suitable for estimating effort of incremental integration testing.

The starting point for the introduction of the $(f_i * f_o)^2$ metric and the LOP * $(f_i * f_o)^2$ metric is the work presented in [7]. Both the metrics are based upon the values that describe how much a unit is connected with other units within a structure chart: fan-in (f_i) and fan-out (f_o). Fan-in gives the number of units that call the unit in the focus of consideration. Fan-out gives the number of units called by the unit in the focus of consideration. The definitions of f_i and f_o cover not only the direct connections between units but the indirect connections, too. The indirect connections are those that use global variables. It has been shown that there is the strong correlation between the number of changes of units and each of the proposed metrics. An increase in each of the metrics leads to an increase in the number of changes. LOC * $(f_i * f_o)^2$ was the second original metric proposed in [7]. A modified version of this metric - LOP * $(f_i * f_o)^2$ is included in our set of metrics suitable for estimating effort of incremental integration testing. The motivation for this inclusion is the same as for LOP.

The quantification of coupling introduced in [3] starts from the number of *m* computed as:

$$m = idc + 2 * icc + odc + 2 * occ + f_o + f_i \quad (1).$$

In this formula, *idc* denotes the number of input data couples, *icc* denotes the number of input control couples, *odc* denotes the number of output data couples, *occ* denotes the number of output control couples, f_o denotes fan-out and f_i denotes fan-in. The data couple computations take care about the direct and indirect connections. The quantification of coupling is defined as C = 1 / m. The higher C corresponds to the better unit from the coupling point of view. Such a unit is easier to modify. The number of *m* is included in the set of measures potentially suitable for estimating effort of incremental integration testing because the effort needed for testing a unit is directly proportional to the *m* number of the unit. We call this *m* number the modified quantification of coupling MCp.

The estimated number of tests is the metric originally proposed for the use in estimating effort of incremental integration testing in [13] and [15]. It is computed as:

$$ENT = 9 * (idc + odc) + 3 * (icc + occ) + f_o + n_{sf} \quad (2).$$

The elements *idc*, *odc*, *icc, occ* and f_o have the same meaning as in (1). The term n_{sf} denotes the number of special features for testing identified in a unit besides functions and interfaces. The

performance of a unit is an example of such a feature. Incremental integration level of testing is divided in phases when the ENT metric is used. Three phases are preparation for testing, the core of testing and reporting on testing. The ENT metric is proposed for estimating effort of the first two phases: preparation for testing and the core of testing. First experimental results with this metric showed the correlation of 0.91 between this metric and actual effort spent in the first two phases of incremental integration testing.

The last metric included in the set of metrics potentially suitable for estimating effort of incremental integration testing is the number of units in a structure chart (NU). This simple metric directly describes the dimension of integration in incremental integration testing. It is proposed as the metric to be used in estimating effort of the third phase of incremental integration testing, reporting on testing, in [13] and [15].

All the enumerated metrics meet the previously defined criteria of selection. LOP and cyclomatic complexity are appropriate for unit testing. The number of units is appropriate for integration testing. From the general point of view, the metrics F2, LF2, MCp and ENT are appropriate for incremental integration testing.

3 Data Collecting

To analyze the applicability of the metrics to estimating effort of incremental integration testing we have developed a number of programs. The programs have been written in C. The development included incremental integration testing. During incremental integration testing the data on the actual effort spent on this testing and the other useful data have been collected. In performing incremental integration testing, a special methodology, the ITeM (Incremental **I**ntegration **Te**sting **M**anagement) methodology [13], has been used. The ITeM methodology is a way of how the various elements necessary for planning and tracking incremental integration testing can be gathered together in one place to direct software engineers during this testing. The brief presentation of doing incremental integration testing when the ITeM methodology is used follows.

The first step to do during incremental integration testing is to plan this task starting from the already created software design. Test planning includes eight steps: 1) determining the features to be tested, 2) identifying the goals of testing, 3) identifying the management constraints, 4) identifying the resources at the disposal, 5) creating the partial plan, 6) estimating, 7) scheduling and 8) planning for the purpose of tracking.

Functions and interfaces of an increment in the focus of testing are the first features identified for testing. Some special features, performance for example, are further identified. Functions are elaborated next and the detailed features for testing are identified. The detailed features are input conditions and their attributes as they are defined in [18]. The input sequence of characters and its length are some examples of input conditions and attributes.

The ITeM methodology allows several different goals of testing to be set for incremental integration testing. The least set of goals of testing requires that the detailed features are covered by tests and that all the problems found in one activity of testing, test planning for example, are resolved before going to the next activity, test design for example. These goals of testing correspond to the completion criteria of testing. The testing task is completed when the goals of testing are met.

The identification of management constraints answers the question if there is a deadline and/or the restricted number of engineers set for the task of testing or there are no special management constraints on time and people required for the task. The identification of resources answers the question of how many software engineers are assigned to the testing task.

The partial plan creation means breaking down the starting task of incremental integration testing into activities across several levels until elementary activities are reached. Each elementary activity is assigned to a single engineer. There are two bases in the partial plan creation:

1. the process model of incremental integration testing defined as a component of the ITeM methodology in [13] and [14] and
2. the design of a program to be tested.

At the first level, the task of incremental integration testing is broken down into three activities that correspond to the phases of the

process model: preparing for testing, the core of testing and reporting on testing. The first activity of preparing for testing is further broken down into three main activities: test planning, test design and coding. Coding is included into the activities of incremental integration testing when the ITeM methodology is used as it is tightly connected with them. In addition, preparing for testing includes the activities of solving problems found in the main activities, problems found in the input software design during test design for example, and the activities of replanning that follow after solving problems. The corrective activities (the activities of solving problems) are included if they are necessary. The places for these activities are reserved in the initial partial plan. That means that finding problems in the input software design is assumed in the initial partial plan creation. This is in accordance with the main aim of testing: finding problems. When we say a problem, we mean any event occurring during testing that requires further investigation. Test planning, test design and coding consider the program to be tested as a whole. If test design is taken, this means that tests for all the increments are designed before coding.

The activity of the core of testing is further broken down into the activities of the same type that correspond to increments. The structure chart of the input software design forms the basis for breaking down this activity into the activities of increments. For each increment, the corresponding core of testing activity includes test implementation (writing a driver for example) and test execution. The current increment must be tested and corrected completely before the core of testing of the next increment begins. The activity of reporting on testing is not further broken down. This activity includes: finding the relationship between the plan and its performance, reporting on problems, forming the data histories and collecting test products for the future use.

In estimating effort of incremental integration testing, the ENT metric is used for the phases of preparing for testing and the core of testing and the NU metric is used for the phase of reporting on testing. The ENT metric is separately computed for all the units first and then summed across the units. Based upon the ENT metric, the NU metric and the corresponding data histories, the effort estimations are computed for the three phases. The effort estimated for preparing for testing is further distributed across the main activities and the corrective and replanning activities, also based upon the data histories. In order to do this distribution, it is necessary to record the effort spent on the separate activities of this phase during the previous testing tasks. The effort estimated for the core of testing is further distributed across the activities of increments proportionally to the ENT metric of each unit.

Starting from the partial plan, the plan's activities and their effort estimations are assigned to days and weeks in scheduling. In the same time, the management constraints are respected as close as possible. Planning for the purpose of tracking includes earned value planning and planning of milestones.

After the plan for the task of incremental integration testing is created in the test planning activity, the task's performance begins following the plan. The tests for all the increments are designed and all the units are coded. If problems are found in the input software design during test planning or test design, they are solved and the input software design is modified appropriately. Then replanning follows. In this way, the input software design is refined before coding. After the phase of preparing for testing is finished, the increment core of testing begins. Finally, after the completion of testing the last increment, the report on the task of incremental integration testing is made.

ITeM is a flexible methodology that offers various options: different goals of testing to be set for a task and different numbers of engineers assigned to the task, among others [13]. The following options of the ITeM methodology have been selected in incremental integration testing the programs considered in this paper:

1. the least set of testing goals has been set,
2. one software engineer has been assigned to the testing task,
3. no management constraints have been defined and
4. the ENT metric and the NU metric have been used in estimating effort of testing.

In addition, the method of equivalence partitioning and the method of boundary value analysis [17] have been used in designing tests for testing functions of units.

The data on the actual effort spent on incremental integration testing the programs have been collected under the previously shown conditions. These data have been originally used to confirm the applicability of the ENT metric and the NU metric to estimating effort of incremental integration testing [15]. It has been shown that the square of the correlation between each of the two metrics and the actual effort spent on the appropriate phase of incremental integration testing is greater than 0.7 and that the likelihood of finding the correlation by chance is less than 0.05.

The same data as in [15] have been used as the starting data in the analysis of the applicability of the metrics introduced in Section 2 to estimating effort of incremental integration testing. The actual effort spent on incremental integration testing includes the actual effort spent on determining the ENT metric and the NU metric among various terms when the ITeM methodology is used. Since the ITeM methodology requires the detailed tracking of performing the testing task, it is possible to separate the actual effort spent on determining these metrics from the rest of the effort. Thus, the actual effort spent on incremental integration testing without the determination of metrics can be obtained. After we computed this value, we determined the other proposed metrics besides ENT and NU for the programs considered and recorded the actual effort spent on the determination of each metric. Then, by summing the actual effort spent on testing without the metric determination and the actual effort spent on determining a particular metric we obtained the actual effort spent on incremental integration testing for each metric.

The differences among the actual efforts spent on incremental integration testing a single program for different metrics are small. They are less than 1.6 %. This result is expectable since the determinations of metrics are similar. To determine each of the metrics it is necessary to go through the program's structure chart and/or the corresponding detailed software design. It is necessary to go through the detailed software design of units presented in PDL to compute LOP and cyclomatic complexity. Then the sum of the metrics of all the units is made. It is necessary to go through the structure chart to compute F2, MCp and NU. At last, it is necessary to go through the structure chart and the detailed design to compute LF2 and ENT.

At last, the task of incremental integration testing has been considered in its completeness in the analysis of the applicability of the metrics to estimating effort of this testing. In other words, the effort spent on all the three phases of incremental integration testing has been taken into account as a single value. The results of this analysis are presented in the next section.

4 Comparison Results

The data on the metrics and the effort spent on incremental integration testing collected on the sequence of programs developed in C are shown in Table 1. The effort is equal to the effective duration of the testing task since one engineer has been assigned to the task. It is given in minutes. Incremental integration testing has been performed in accordance with the ITeM methodology and under the conditions presented in the previous section. Thus, the effort presented in Table 1 include the determination of the ENT metric and the NU metric. The efforts for the other metrics are not given although they are computed and used in the correlation computations. They are not given since they are very similar to those shown in Table 1. For example, for the program named Cyclomatic Complexity Meter the effort goes from 1448 minutes for the NU metric to 1470 minutes for the LF2 metric. In addition, the formulas given in Section 2 have been used in computing the metrics of separate units in increments. Then the sum of the metric numbers across all the increments has been created for each metric. The exception is the NU metric that corresponds to the number of units of a structure chart.

To analyze the applicability of the metrics to estimating effort of incremental integration testing the correlation between a particular metric and the actual effort spent on this testing has been computed as well as the likelihood of finding the correlation by chance. The results are shown in Table 2. Correlation represents the measure of the linear relationship between two variables. If the existence of the linear relationship is confirmed, it is possible to estimate the second variable based upon the first variable and this relationship. This analysis of the applicability of metrics to

Table 1 Metrics and the effort spent on incremental integration testing collected on a number of programs

Program	Longest Line	Calculator	Triangle	Empty Line Counter	Line Counter	Cyclomatic Complexity Meter	Expert System User Interface *
Effort	296	2787	795	1572	1562	1459	4262
LOP	20	279	211	310	352	356	1317
CC	3	43	45	66	61	45	194
F2	16	71	53	141	147	120	1496
LF2	320	4361	3899	10903	13510	9874	48307
MCp	5	59	34	58	64	58	338
ENT	34	287	158	226	280	337	842
NU	1	8	5	8	7	7	43
* Presented in [1] and [12]							

estimating effort of incremental integration testing is done in accordance with the suggestions given for PSP [9]. If the square of the correlation of two variables is greater than 0.7 or greater, and the likelihood of finding the correlation by chance is less than 0.05 or less, then the metric is suitable for estimating effort [9].

Table 2 shows that the best effort estimation of incremental integration testing is obtained when the ENT metric is used. It can be expected since the ENT metric has originally been proposed for the estimation purpose [15]. The other metrics can be divided into three groups. LOP, NU and MCp form the first group. They give the similar squares of the correlation: 0.786, 0.780 and 0.780 respectively. Among these metrics, the NU metric can be computed in the simplest way. CC forms the second group and it has the square of the correlation of 0.755. F2 and LF2 form the third group with the square of the correlation around the limit of 0.7. It can be concluded that the ENT metric should be used to obtain the best effort estimation for incremental integration testing. Good results could be obtained with the NU metric, too. This metric is especially acceptable when we want to eliminate the excessive computations.

5 Conclusion

The applicability of seven design metrics to estimating effort of incremental integration testing is analyzed in this paper. These metrics are selected because they are potentially suitable for this estimation as is shown in Section 2. The data collected on the sequence of programs developed in C under the conditions defined in Section 3 have been used in this analysis. They show that the best estimation is obtained with the ENT metric. Then LOP, NU and MCp follow with the similar estimation capabilities. Among these metrics, the NU metric is computed most easily.

Table 2 Correlations and significance values

Metric	Correlation	Square of the correlation	Likelihood of finding the correlation by chance
LOP	0,886	0,786	0,00259
CC	0,869	0,755	0,00388
F2	0,832	0,692	0,00765
LF2	0,842	0,709	0,00650
MCp	0,883	0,780	0,00279
ENT	0,921	0,848	0,00092
NU	0,883	0,780	0,00280

When the most precise estimation for incremental integration testing is needed, the ENT metric should be used. When an acceptable estimation and the minimum computations are needed, the NU metric should be used. At least, this conclusion can be accepted under the conditions shown in Section 3 and for the programs given in Table 1. Even if the conditions are not met, the proposed metrics should be used, as it is a better solution than to have no estimation at all.

References:

[1] N. Afgan, P.M. Radovanović and A.G. Blokh, An Expert System for Boiler Surface Fouling Assessment, In Hanjalić, K. and Kim, J.H.(Eds), *Expert Systems and Simulations in Energy Engineering*, Begel

House, New York, USA, 1995, pp. 219-224.

[2] S.R.Dalal and A.A. McIntosh, When to Stop Testing for Large Software Systems with Changing Code, *IEEE Trans. on Software Engineering*, Vol. 20, No. 4, 1994, pp. 318-323.

[3] H. Dhama, Quantitative Models of Cohesion and Coupling in Software, *Journal of Systems and Software*, Vol. 29, No. 4, 1995, pp. 65-74.

[4] C. Eastel and G. Davies, *Software Engineering: Analysis and Design*, McGraw-Hill, London, UK, 1989.

[5] W. Ehrlich, B. Prasanna, J. Stampherl and J. Wu, Determining the Cost of a Stop-Test Decision, *IEEE Software*, Vol. 10, No. 2, 1993, pp. 33-42.

[6] D. Gelperin and B. Hetzel, "The Growth of Software Testing", *Communications of the ACM*, Vol. 31, No. 6, pp. 687-695.

[7] S. Henry and D. Kafura, Software Structure Metrics Based on Information Flow, *IEEE Trans. on Software Engineering*, Vol. SE-7, No. 5, 1981, pp. 510-518.

[8] B. Hetzel, *The Complete Guide to Software Testing*, 2nd ed., Wiley, New York, USA, 1988.

[9] W.S. Humphrey, *A Discipline for Software Engineering*, Addison-Wesley, Reading, Massachusetts, USA, 1995.

[10] T.J. McCabe, A Complexity Measure, *IEEE Trans. on Software Engineering*, Vol. SE-2, No. 4, 1976, pp. 308-320.

[11] T.J. McCabe and C.W. Butler, Design Complexity Measurement and Testing, *Communications of the ACM*, Vol. 32, No. 12, 1989, pp. 1415-1425.

[12] Z. Mihajlović, An Expert System for Monitoring of Fouling in a Power Boiler, *Proceedings of the 41st Yugoslav Conference of ETRAN*, Zlatibor, Yugoslavia, Vol. 3, 1997, pp. 239 - 242 (in Serbian).

[13] Z. Mihajlović, Knowledge-Based Planning and Tracking the Level of Unit and Integration Software Testing, Ph.D. Dissertation, University of Belgrade, Belgrade, Serbia and Montenegro, 2003.

[14] Z. Mihajlović, D. Velašević and N. Mastorakis, A Preventive Process Model of the Incremental Integration Level of Software Testing, *WSEAS Trans. on Computers*, Vol. 2, Issue 1, 2003, pp. 30-35.

[15] Z. Mihajlović and D. Velašević, Measures for Estimating Effort of Incremental Integration Software Testing, *Proceedings of the International Conference on Computer, Control and Communication Technologies CCCT'03*, Orlando, USA, Vol. I, 2003, pp. 434-439.

[16] J.D. Musa and A.F. Ackerman, Quantifying Software Validation: When to Stop Testing? *IEEE Software*, Vol. 6, No. 3,1989, pp. 19-27.

[17] G.J. Myers, *The Art of Software Testing*. Wiley, New York, USA, 1979.

[18] T.J. Ostrand and M.J. Balcer, The Category-Partition Method for Specifying and Generating Functional Tests, *Communications of the ACM*, Vol. 31, No. 6, 1988, pp. 676-686.

[19] W. Perry, *Effective Methods for Software Testing*, Wiley, New York, USA, 1995.

[20] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, 4th edition, McGraw-Hill, New York, USA, 1997.

[21] N.D. Singpurwalla, Determining the Optimal Time Interval for Testing and Debugging Software, *IEEE Trans. on Software Engineering*, Vol. 17, No. 4, 1991, pp. 313-319.

[22] J. Su and P.R. Ritter, Experience in Testing the Motif Interface, *IEEE Software*, Vol. 8, No. 2, 1991, pp. 26-33.

[23] J. Voas, L. Morell and K. Miller, Predicting Where Faults Can Hide from Testing, *IEEE Software*, Vol. 8, No. 2, 1991, pp. 41-48.

[24] J.A. Whittaker, What Is Software Testing? And Why Is It So Hard?, *IEEE Software*, Vol. 17, No. 1, 2000, pp. 70-79.

[25] T. Yamaura, How to Design Practical Test Cases, *IEEE Software*, Vol. 15, No. 6, 1998, pp. 30-36.