Minimising the Preparation Cost of Runtime Testing based on Testability Metrics

Alberto Gonzalez-Sanchez Éric Piel Hans-Gerhard Gross Arjan J.C. van Gemund
Department of Software Technology, Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
Email: {a.gonzalezsanchez,e.a.b.piel,h.g.gross,a.j.c.vangemund}@tudelft.nl

***Abstract*—Test cost minimisation approaches have traditionally been devoted to minimising "execution costs", while maximising coverage or reliability. However, in a runtime testing context, the amount of coverage or reliability that can be achieved, in other words, the system's Runtime Testability, is limited by the adverse effects that the interferences of runtime tests have on the system. Supporting runtime testing, therefore, introduces an additional cost in "preparatory" activities in software, (e.g., testable components) and in hardware (e.g., more memory), before certain runtime tests can be executed.**

In this paper we present a low-complexity, cost minimisation algorithm for the optimal selection of preparation activities, based on a near-optimal trade-off between preparation cost and a structure-based measurement of Runtime Testability, coined the Runtime Testability Metric (RTM). We perform a theoretical and empirical validation of RTM, showing that RTM is indeed a valid, and reasonably accurate measurement with ratio scale. We also present empirical data demonstrating the near-optimal performance at a low computational cost of our algorithm.

I. Introduction

Critical and high-availability systems, such as air traffic control systems, systems of the emergency units, and banking applications, are becoming more and more complex and dynamic. Moreover, in the case of Systems of Systems, or Service Oriented Architectures components may not be available until deployment time, e.g., third party external services. Components can be even unknown at deployment time.

The testing process of these kind of systems was traditionally performed either in a separate, identical copy of the system, or by taking the system offline. This cannot be done anymore for modern systems such as the ones already mentioned [3], [6], [11].

Runtime testing is emerging as the solution for the validation and acceptance testing of the above systems. Runtime testing is a testing method that has to be carried out in the final execution environment of a system [3], [4], [9], [15], [17].

However, in practice many parts of the system cannot be tested because they would affect the system in critical ways that are difficult to control or impossible to recover from. In order to test those parts, an additional expense in the test *preparation* phase is required, both in software development and in hardware. Therefore, not only the execution costs of testing, but also the preparation costs need to be minimised, by means of a *preparation plan*: an optimal combination of modules of the system that have to be made runtime testable.

The optimisation problem of preparation cost with respect to *runtime testability* has only been addressed very recently [10], [12]. Runtime testability (RTM) constitutes an upper bound on what can be subsequently achieved in the test *execution* phase based on the systems's structure and the location of untestable components.

The previous definition of RTM had an umber of shortcomings: (1) it did not take into account the real-valued preparation costs; (2) no theoretical characterisation of RTM was performed from the point of view of measurement theory; (3) no empirical evaluation of RTM was performed; and (4) no computationally efficient algorithm was presented that would allow test engineers to efficiently generate a preparation plan for systems of realistic size.

In this paper we address the above issues. In particular, this paper makes the following contributions:

1) We present a new, improved model and definition of RTM.
2) We provide a measurement-theoretical characterisation of RTM.
3) We perform an empirical evaluation of RTM based on two real systems.
4) We introduce two low-cost greedy algorithms to be used in obtaining a near-optimal preparation plan.

The paper is organised as follows. Section II introduces RTM. In Section III its theoretical characterisation is performed. The empirical study on the accuracy of RTM is presented in Section IV. The low-cost prioritisation algorithm is presented in Section V. Related work is presented on Section VI. Section VII wraps up the paper and addresses future work.

II. Runtime Testability

Runtime testability is the degree to which a system can be runtime tested. A numerical measurement for the runtime testability of a system can be defined in terms of what amount of fault sites in the system can be runtime tested, relative to the maximum test coverage attainable by the system testers under runtime testing conditions.

Runtime testability can be expressed empirically by the maximum coverage that can be reached by runtime tests, defined as

$$ERT = |Cov| \tag{1}$$

where Cov represents the set of possible fault sites (according to a coverage criterion) which can be covered by runtime tests. The rationale behind this definition is that if a fault site cannot be exercised in testing, the number of faults that remain undetected in the system after runtime testing will be higher.

In this section we will present the formal definition of RTM and the prediction model used to obtain it.

A. Model of the System

We use a dependency graph, which captures the architecture of the system at the moment of testing. This way of representing dependencies is known as Component Interaction Graph (CIG) [19].

A CIG is defined as a directed graph with weighted vertices, $CIG = \langle V, V_0, E, c \rangle$, where

- $V \equiv V_P \cup V_R$: vertices in the graph, formed by the union of the sets of provided and required operations by the components' interfaces.
- $V_0 \subseteq V$: input operations to the system, i.e., operations directly accessible to test scripts.
- $E \subseteq V \times V$: edges in the graph, representing dependencies between operations in the system. E.g., if $(v_1, v_2) \in E$, v_1 depends on v_2.
- $c : V \to \mathbb{R}^+$: function that maps a specific operation to the preparation cost that is going to be optimised.

Because of the lack of state and control flow information in the CIG model, the following assumptions are made about the actual behaviour of the system:

1) Control flow paths are independent.
2) When the control flow reaches an operation (vertex) it will always propagate to all dependencies (edges).

The first assumption is needed because the CIG does not reflect data dependencies between different operations. The second assumption is necessary because we do not know which edges in the CIG will be traversed by a test case. In the extreme case, the interaction might propagate through all edges, affecting all reachable vertices.

B. Definition of RTM

Following our model and assumptions, the total preparation cost needed to involve an operation in a runtime test, taking into account the individual preparation costs of all the operations it depends on, is defined as

$$C(v_i) = \sum_{v_j \in S_{v_i}} c(v_j) \tag{2}$$

where v_i and v_j are operations, and S_{v_i} is the set of successors of vertex v_i, i.e., all the vertices reachable from v_i including v_i itself.

The initial definition of RTMconsidered only those operations where $C(v_i) = 0$, defining RTM as

$$RTM_{old} = |\{v_i \in V : C(v_i) = 0\}| \tag{3}$$

This definition assumed that any operation on any interface of any component in the system can be invoked directly by test scripts, i.e., that every operation is an input operation. This is unrealistic in most cases. In this paper we drop this assumption by means of the set of possible input vertices V_0, by requiring that operations can be reached from a testable input vertex $v_0 \in V_0$, i.e., that $v_i \in S_{v_0}$ and whose $C(v_0) = 0$. RTM can be then defined as

$$\begin{aligned} RTM = |\{&v_i \in V : C(v_i) = 0 \wedge \\ &\exists v_0 \in V_0 : v_i \in S_{v_0} \wedge C(v_0) = 0\}| \end{aligned} \tag{4}$$

III. Theoretical Validation

In this section, we establish the characteristics of the RTM measurement from a measurement-theoretical point of view. It allows us to identify what statements and mathematical operations involving the metric and the systems it measures are meaningful and consistent. We will concentrate (1) on RTM's fundamental properties, and (2) on its type of scale.

A. Fundamental Properties

In this section, we will study the properties required for any measurement. The properties were described by Shepperd and Ince in [16] through an axiomatic approach.

Axiom 1: It must be possible to describe the rules governing the measurement.

This is satisfied by the formal definition of RTM and the CIG.

Axiom 2: The measure must generate at least two equivalence classes.

$$\exists p, q \in CIG : RTM(p) \neq RTM(q)$$

It is trival to obtain two CIGs that satisfy this axiom and no example will be provided.

Axiom 3: An equality relation is required.

This axiom is satisfied given that our measurement is based on natural numbers, for which an equality relation is defined.

Axiom 4: There must exist two or more structures that will be assigned the same equivalence class.

$$\exists p, q \in CIG : RTM(p) = RTM(q)$$

It is trival to obtain two CIGs that satisfy this axiom and no example will be provided due to space concerns.

Axiom 5: The metric must preserve the order created by the empirical property it intends to measure. This axiom is also known as the Representation Theorem.

$$\forall p, q \in CIG : p \underset{rt}{\succeq} q \Leftrightarrow RTM(p) \geq RTM(q)$$

where $\succeq_{rt}$ represents the empirical relation 'more runtime testable than'.

This last axiom means that for any two systems, the ordering produced by the empirical property "runtime testability" has to be preserved by RTM. It is possible to find systems for which this axiom does not hold for RTM, because of the assumptions that had to be made (see Section II-A). However, we can empirically assess the effect of these assumptions on the consistency and accuracy of RTM. An empirical study about the accuracy of RTM is presented in Section IV.

B. Type of Scale

The theoretical characterisation of the metric's scale type (i.e., ordinal, interval, ratio, absolute) determines which mathematical and statistical operations are meaningful. This is important, because certain optimisation algorithms require specific mathematical operations that might not be meaningful for RTM, for example for defining heuristics as we will see in Section V.

Assuming RTM satisfies Axiom 5, i.e., it preserves the empirical ordering of runtime testability, then, by definition, RTM defines a homomorphism from runtime testability to the Natural numbers. Therefore, by the ordered nature of Natural numbers, we can assert that RTM can be used as an *ordinal scale* of measurement. As will be noted in Section IV, in practice this is true only for systems in which our assumptions about control flow and dependencies hold.

In order to be able to use RTM as a ratio scale measurement, in addition to the requirements for the ordinal type of scale being satisfied, a concatenation operation with an additive combination rule [20] must exist. A meaningful concatenation operation is creating the union of both systems by disjoint union of their CIG models. This operation, $\cup : CIG \times CIG \rightarrow CIG$, can be defined as $A \cup B = \langle V, V_0, E, c \rangle$, where

- $V \equiv V_A \cup V_B$
- $V_0 \equiv V_{0A} \cup V_{0B}$
- $E \equiv E_A \cup E_B$
- $c(v) = \begin{cases} c_A(v) & \text{if } v \in V_A \\ c_B(v) & \text{if } v \in V_B \end{cases}$

For this concatenation rule, the additive combination rule

$$RTM(A \cup B) = RTM(A) + RTM(B) \tag{5}$$

can be used. Therefore, RTM can be used as a ratio scale with extensive structure (e.g., like mass or length), with respect to the disjoint union operation.

C. Summary

We will highlight the most relevant practical implications of the theoretical properties of RTM obtained in the previous section.

Because we proved that RTM fulfils the minimal properties of any measurement, RTM can be used to *discriminate* and *equalise* systems. Therefore, the statements '*system A has a different runtime testability than B*', and '*systems A and B have the same runtime testability*', are meaningful. Moreover, as we proved RTM has an ordinal scale type, RTM can be used to *rank* systems. The statement '*system A has more runtime testable operations than B*' becomes meaningful, and this enables us to calculate the median of a sample of systems, and Spearman's rank correlation coefficient.

Furthermore, by proving the ratio scale for RTM, it can also be used to *rate* systems, making the statement '*system A has X times more runtime testable operations than B*' a meaningful one. This allows performing a broad range of statistic operations meaningfully, including mean, variance, and Pearson's correlation coefficient.

RTM can also be used alone to reason about the composition of two systems. Due to its additive combination rule, '*systems A and B composed, will be* $RTM(A) + RTM(B)$ *runtime testable*' is a meaningful statement, provided that A and B are disjoint.

IV. Empirical Validation

In this section we conduct a number of experiments in order to empirically determine how accurate RTM is with respect to the empirical property of "runtime testability" (ERT). Two systems were used in the experiment: (1) AISPlot, a system-of-systems taken from a case study in the maritime safety and security domain, and (2) WifiLounge, an airport's wireless access-point system [5].

A. Experimental Setup

To obtain the value of RTM, vertices are first classified into testable and untestable by means of $C(v)$ (see Eq. 2). Our goal is to assess the influence of the assumptions made when defining RTM, in the number of false positives and false negatives of this classification, and in the final value of RTM.

In order to have a baseline for comparison, the naive approach of just counting directly testable operations was used, in addition to RTM_{old} and RTM, 500 variations of the AISPlot and WifiLounge systems with different RTM values were generated by choosing the untestable vertices by randomly sampling in groups of increasing size from 2 to 30 untestable vertices. The value of ERT to perform the comparison was obtained by executing an exhaustive test suite in terms of vertices and execution paths. The set of operations covered when executing each test case was recorded. If a test case used any untestable operation, none of the operations covered by the test were counted.

B. Results

From each system and metric pair in this experiment, we recorded the following data:

- M_{set}: Set of operations classified as testable.

- Cov: Set of operations covered.
- $f_p = (|M_{set} - Cov|)/|M_{set}|$: false positive rate, i.e., operations wrongly classified as testable.
- $f_n = (|Cov - M_{set}|)/|Cov|$: false negative rate, i.e., operations wrongly classified as untestable.
- $\bar{e} = ||M_{set}| - |Cov||$: absolute error between the predicted and empirical runtime testabilities.

	System	f_p	f_n	$\bar{e}$
NTES	AISPlot	0.942	0.000	83.487
	WifiLounge	0.713	0.000	57.093
RTM_{old}	AISPlot	0.882	0.107	15.012
	WifiLounge	0.577	0.079	27.664
RTM	AISPlot	0.411	0.111	2.418
	WifiLounge	0.306	0.128	9.101

Table I
FALSE POSITIVE/NEGATIVE RATE, AND ERROR

Table I shows the rates of false positives and false negatives, along with the absolute error, averaged over 500 runs. The deviation between RTM and the actual covered operations for each sample can be seen in the three plots in Figure 1. The dashed line represents the ideal target. Any point above it, constitutes an overestimation error, and below it, an underestimation error.

NTES: It can be seen that NTES has an extremely high error caused by its high false positive rate (94% and 71%). NTES has no false negatives as it classifies as untestable only the vertices which are directly untestable, disregarding dependencies.

RTM_{old}: By taking control flow dependencies into account, the false positive rate of RTM_{old} is lower than NTES, at the price of introducing a number of false negatives. False negatives appear because in some cases where the control flow does not propagate to all of the operations' dependencies, as we had assumed. Still, because of the assumptions that test interactions can start in any vertex, and that the test paths are independent, the number of false positives is considerable. The number of input vertices in WifiLounge is proportionally higher than for AISPlot. Hence, the number of false positives caused by this assumption is lower.

RTM: By taking input vertices into account, the amount of overestimation decreases dramatically for both systems. Still, the false positive rate is significant. Therefore, we conclude that assuming that paths are not dependent is not very reasonable and needs to be addressed in future work.

The increase of false negatives makes more apparent the consequences of the assumption that control flow is always being transmitted to dependencies. The error caused by this assumption is augmented by the fact that it also applies to the input paths to reach the vertex being considered.

V. PREPARATION PLAN COST MINIMISATION

Once the theoretical and empirical properties of RTM have been established, RTM can be safely used for the

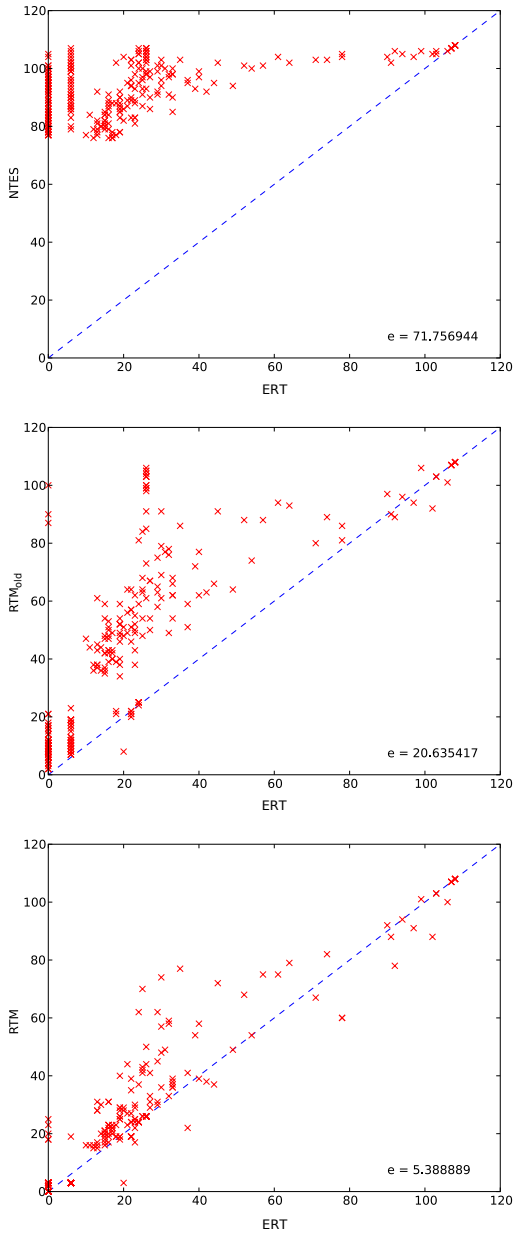

Figure 1. Accuracy of NTES, RTM_{old} and RTM

elaboration of an optimal preparation plan with the objective of achieving the highest possible runtime testability within a given budget b. Mathematically, this problem is formulated as

$$\begin{aligned} &\text{maximise:} && RTM \\ &\text{subject to:} && \sum c(v_j) \cdot x_j \leq b, \quad x_j \in \{0,1\} \end{aligned}$$

where b is the maximum budget available, and x_j represents the decision of including vertex v_j in the preparation plan or not. Unfortunately, combinatorial optimisation problems

such as this one are NP-Hard, therefore, an optimal preparation plan cannot be calculated in polynomial time. An approximate solution with a smaller computational complexity is needed.

A. Near-optimal Minimisation

In this section we present an approach to generate an approximate preparation plan by using a greedy selection heuristic method.

1) Heuristics: First, we consider a heuristic that takes a pessimistic approach: it ranks higher the vertices that provide the highest gain on testability in the next step. The count is divided by the cost to penalise expensive nodes. The heuristic is defined as

$$h_{pessimistic}(v_i) = \frac{1}{c_i}(RTM_{v_i} - RTM) \tag{6}$$

where RTM_{v_i} is the value of RTM after the costs of vertex v_i have been covered. Given the pessimistic nature of this heuristic, we expect this heuristic rank to perform well for low budgets, and poorly for higher ones.

Our second heuristic takes an optimistic approach. It ranks higher the vertices that appear in the highest number of P sets, i.e., the vertices that will fix the most uncoverable vertices assuming they only depend on the vertex being ranked. This value is also divided by the cost to penalise expensive nodes over cheaper ones. The heuristic is defined as

$$h_{optimistic}(v_i) = \frac{1}{c_i}|\{v_j \mid v_i \in S_{v_j}\}| \tag{7}$$

By ignoring the fact that an uncoverable vertex may be caused by more than one untestable vertex, and that the vertex may not be reachable through a testable path, this second heuristic will take very optimistic decisions on the first passes which affect the quality of results for proportionally low budgets, and yields a better performance for higher budgets.

The optimistic heuristic will skip many low-cost solutions (its curve being much lower than the optimum), while the pessimistic heuristic is more precise for low costs but completely misses good solutions with higher budgets. A simple solution to address these shortcoming is to combine both heuristic rankings by taking the best results of both.

B. Computational Complexity and Error

The time complexity of the preparation plan computation depends on the time complexity of the heuristic function. As in each pass there is one less vertex in U, the H function is evaluated $|U|, |U|-1, \ldots, 1$ times while searching for the maximum. In total, it is evaluated $\frac{|U|^2}{2}$ times.

Both heuristic functions perform a sum that depends on the total number of vertices in the system, the complexity of the PREPARATIONPLAN function is $O(|V| \cdot |U|^2)$, and therefore polynomial.

Although a polynomial complexity is much more appealing than the $O(2^{|U|})$ complexity of the exhaustive search, the approximation error has to be taken into account.

A number of experiments were conducted to evaluate the approximation error of our heuristic method. The plot in Figure 2 shows the evolution of the relative average approximation error of RTM for each of our heuristics, as a function of the number of untestable operations $|U|$. The optimal solution function is obtained by exhaustive search.

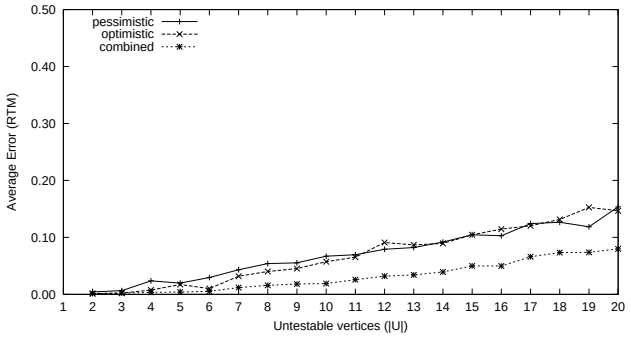

Figure 2. Performance of the approximate algorithms compared to the optimal

The average error incurred by our heuristics is very low considering the processing time required for their calculation. Combining the rankings created by both the long and short-term heuristics, by choosing the maximum of either solution, yields the best of both methods while maintaining the low computational complexity. This can be seen in the $combined$ error plot in Figure 2.

VI. RELATED WORK

Testability and testing effort was originally related to the probability of a fault causing an error, this error propagating to the output [13], [18], and the error being actually detected by the test oracle [1].

However, testability is affected by many other factors and its measurement has been studied extensively from various points of view. A descriptive model of testability is presented in [2]. A second descriptive model of testability is presented in [8]. No evaluation, either theoretical nor empirical, is performed. A number of potential quantitative metrics for testability, relating system structure and dependencies to regression test cost, were studied in [14]. A metric of testability related is proposed in [7], based on the idea that if a module's input and output domains are not completely defined by the module's interface contract, it will require a higher test effort (i.e., time needed to define test cases).

VII. CONCLUSIONS AND FUTURE WORK

In this paper we have performed an in-depth study, theoretical and empirical, of RTM, a metric designed to measure the effect that untestable operations will have on

the runtime testability of a system. RTM has the properties of a ratio scale type, and a relatively good accuracy although there is room for improvement given the strenght of the assumptions it relies on.

We have provided a low-cost approximation algorithm to the improvement of the system's runtime testability which computes near-optimal preparation plans, significantly reducing the computation time. This allows system engineers to elaborate a preparation plan to prioritise the expenses in the preparation phase of testing, with the goal of increasing the runtime testability of the system at a minimum cost.

Future work will address the assumptions made in the system model used to obtain RTM and increase the sample of systems used in the empirical evaluation of RTM.

ACKNOWLEDGMENTS

The authors wish to thank their partners in the Poseidon project in the Embedded Systems Institute (ESI), Eindhoven, The Netherlands. This project is partially supported by the Dutch Ministry of Economic Affairs under the BSIK03021 program.

REFERENCES

[1] A. Bertolino and L. Strigini. Using testability measures for dependability assessment. In *ICSE '95: Proceedings of the 17th international conference on Software engineering*, pages 61–70, New York, NY, USA, 1995. ACM.

[2] R. V. Binder. Design for testability in object-oriented systems. *Communications of the ACM*, 37(9):87–101, 1994.

[3] D. Brenner, C. Atkinson, O. Hummel, and D. Stoll. Strategies for the run-time testing of third party web services. In *SOCA '07: Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications*, pages 114–121, Washington, DC, USA, 2007. IEEE Computer Society.

[4] D. Brenner, C. Atkinson, R. Malaka, M. Merdes, B. Paech, and D. Suliman. Reducing verification effort in component-based software engineering through built-in testing. *Information Systems Frontiers*, 9(2-3):151–162, 2007.

[5] T. Bures. Fractal bpc demo.

[6] T. Dumitras, F. Eliassen, K. Geihs, H. Muccini, A. Polini, and T. Ungerer. Testing run-time evolving systems. Number 09201 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl.

[7] R. S. Freedman. Testability of software components. *IEEE Transactions on Software Engineering*, 17(6):553–564, 1991.

[8] J. Gao and M.-C. Shih. A component testability model for verification and measurement. In *COMPSAC '05: Proceedings of the 29th Annual International Computer Software and Applications Conference*, volume 2, pages 211–218, Washington, DC, USA, 2005. IEEE Computer Society.

[9] A. González, É. Piel, and H.-G. Gross. Architecture support for runtime integration and verification of component-based systems of systems. In *1st International Workshop on Automated Engineering of Autonomous and run-time evolving Systems (ARAMIS 2008)*, pages 41–48, L'Aquila, Italy, Sept. 2008. IEEE Computer Society.

[10] A. González, E. Piel, and H.-G. Gross. A model for the measurement of the runtime testability of component-based systems. In *Software Testing Verification and Validation Workshop, IEEE International Conference on*, pages 19–28, Denver, CO, USA, 2009. IEEE Computer Society.

[11] A. González, É. Piel, H.-G. Gross, and M. Glandrup. Testing challenges of maritime safety and security systems-of-systems. In *Testing: Academic and Industry Conference - Practice And Research Techniques*, pages 35–39, Windsor, United Kingdom, Aug. 2008. IEEE Computer Society.

[12] A. Gonzalez-Sanchez, É. Piel, and H.-G. Gross. RiTMO: A method for runtime testability measurement and optimisation. In *Quality Software, 9th International Conference on*, Jeju, South Korea, Aug. 2009. IEEE Reliability Society.

[13] D. Hamlet and J. Voas. Faults on its sleeve: amplifying software reliability testing. *SIGSOFT Software Engineering Notes*, 18(3):89–98, 1993.

[14] S. Jungmayr. Identifying test-critical dependencies. In *ICSM '02: Proceedings of the International Conference on Software Maintenance (ICSM'02)*, pages 404–413, Washington, DC, USA, 2002. IEEE Computer Society.

[15] C. Murpy, G. Kaiser, I. Vo, and M. Chu. Quality assurance of software applications using the in vivo testing approach. In *ICST '09: Proceedings of the 2nd international Conference on Software Testing*. IEEE Computer Society, 2009.

[16] M. Shepperd and D. Ince. *Derivation and Validation of Software Metrics*. Oxford University Press, 1993.

[17] D. Suliman, B. Paech, L. Borner, C. Atkinson, D. Brenner, M. Merdes, and R. Malaka. The MORABIT approach to runtime component testing. In *30th Annual International Computer Software and Applications Conference*, pages 171–176, Sept. 2006.

[18] J. Voas, L. Morrel, and K. Miller. Predicting where faults can hide from testing. *IEEE Software*, 8(2):41–48, 1991.

[19] Y. Wu, D. Pan, and M.-H. Chen. Techniques for testing component-based software. In *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems*, pages 222–232, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[20] H. Zuse. *A Framework of software measurement*. de Gruyter, 2000.