A Diagnostic Point of View for the Optimization of Preparation Costs in Runtime Testing

Alberto Gonzalez-Sanchez Éric Piel Hans-Gerhard Gross Arjan J.C. van Gemund

Delft University of Technology, Software Technology Department
Mekelweg 4, 2628 CD Delft, The Netherlands
Email: {a.gonzalezsanchez,e.a.b.piel,h.g.gross,a.j.c.vangemund}@tudelft.nl

***Abstract*—Runtime testing is emerging as the solution for the validation and acceptance testing of service-oriented systems, where many services are external to the organization, and duplicating the system's components and their context is too complex, if possible at all. In order to perform runtime tests, an additional expense in the test preparation phase is required, both in software development and in hardware.**

Preparation cost prioritization methods have been based on runtime testability (i.e, coverage) and do not consider whether a good runtime testability is sufficient for a good runtime diagnosis quality in case faults are detected, and whether this diagnosis will be obtained efficiently (i.e., with a low number of test cases). In this paper we show (1) the direct relationship between testability and diagnosis quality, that (2) these two properties do not guarantee an efficient diagnosis, and (3) a measurement that ensures better prediction of efficiency.

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. The number and complexity of the components that form the systems is growing. 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 [5], [9], [15]. In a service-oriented system where many services are external to the organization, duplicating the system's components and their context is too complex, if possible at all. Taking the system offline is also not possible given the high availability requirements of such systems.

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. It has been proposed for component-based systems [6], [13], [30], web services [5], and for Java applications [26].

In practice many parts of the system cannot be tested, because the tests would affect the system in critical ways that are difficult to control or impossible to recover from. For instance, firing a missile while testing part of a combat system. In order to test those parts, an additional expense in the test *preparation* phase is required, both in software development, (e.g., test wrappers, testable components, simulators) and in hardware (e.g., an extra processor, more memory, higher capacity data links). Therefore, not only the execution costs of testing, but also the preparation costs, need to be optimized, i.e., prioritized.

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	C_t	o_i
t_1	1	0	1	0	0	1	1	0	10	1
t_2	0	1	0	1	0	0	0	0	10	0
t_3	0	0	1	0	1	0	1	0	10	0
t_4	0	1	0	0	0	0	0	1	10	0
t_5	1	0	0	0	1	0	0	0	10	1
t_6	0	0	0	0	0	1	1	0	10	0
t_7	0	1	0	1	0	1	0	1	10	0
t_8	0	1	1	0	1	0	1	1	10	0
t_9	1	0	1	0	0	1	0	1	10	1
C_p	0	0	0	0	0	0	10	10		

TABLE I
EXAMPLE COVERAGE MATRIX, EXECUTION, AND PREPARATION COSTS

We can represent test cases and their coverage in a matrix with N rows and M columns. Columns correspond to components, and rows to tests. This is shown in Figure I. If test case t_i covers component c_j, this is represented with element $a_{ij} = 1$ in the test matrix. Otherwise, $a_{ij} = 0$. The execution costs are represented by the C_t column, whereas the preparation costs associated to components are represented by the C_p row.

There exists extensive literature on prioritizing the execution costs of a test suite. Test execution prioritization techniques operate on the vertical dimension of the matrix, selecting test cases based on their execution cost and their utility. This utility can be a measurement of either fault detection [10], [22], [25], [28], [29], [32] or fault localization [17] potential. On the other hand, preparation cost optimization methods operate on the horizontal dimension, and select components to prepare based on runtime testability measurements [16], [18].

However, preparation cost prioritization methods fall short in two respects. First, one must consider that testing is very often paired to a diagnosis phase. Since runtime testability is

optimized for coverage, a question left unanswered is whether a good runtime testability is sufficient for a good runtime diagnosis quality. Second, different preparations will enable the execution of different test cases, which means that, in practice, test preparation prioritization should take into account the test execution cost of the tests it is 'enabling'. As important as the final coverage or diagnosis quality of the test cases, is the rate at which such coverage or diagnosis quality evolves.

In this paper we address these issues, showing (1) the direct relationship between testability and diagnosability, (2) that these two measurements do not guarantee an efficient diagnosis, and (3) a third measurement that ensures better prediction of efficiency.

The paper is organized as follows. Section II presents the concepts of runtime testing. Section III introduces the necessary concepts of fault diagnosis. Section IV introduces the concept of diagnosis efficiency and measurement. Section V presents the systems used in our evaluation and our empirical results. Section VI positions this work with respect to related work. Finally, Section VII wraps up the paper and presents direction for further research.

II. Runtime Testing

One of the major challenges of runtime testing is the interference that it will cause on the system's state or resource availability. In the worst case, runtime tests will affect the system's environment in critical ways that are difficult to control or impossible to recover from, e.g., firing a missile while testing part of a combat system.

Runtime testing is significantly influenced by two main characteristics of the system: *test sensitivity*, and *test isolation* [14], [30]. Test sensitivity characterizes which fraction of the features of the system will cause interference between the running system and the test operations, e.g., a component having internal state, a component's internal/external interactions, resource limitations, etc. Conversely, test isolation techniques are applied by engineers to specific components to counter the test sensitivity, e.g., state duplication or component cloning, usage of simulators, resource monitoring, etc.

A. Runtime Testability

Runtime testability is the degree to which a system can be runtime tested [14]. 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. Previous work in this area has tried to estimate runtime testability by using a graph model of the system [14], [18]. In this paper we will use a more direct approach based on a the existing test suite for the system.

To calculate the runtime testability of a given system we will use the coverage matrix of the system under tests, such as the one in Table I.

The total preparation cost needed to execute a test (not counting its execution costs) is the sum of all the preparation costs of the operations covered by the test

$$C_p(t_i) = \sum_{j=1}^{M} a_{ij} \cdot c(c_j)$$

Where M is the nuymber of components in the system. This cost, however, does not concern test t_i exclusively. Since multiple tests may cover the same component, these costs are shared between all the multiple tests that cover the component.

The runtime testability metric can be calculated based on the coverage matrix of the system as the coverage level reachable by the tests of cost $C_p = 0$, as

$$RTM = \frac{1}{M} \cdot \sum_{j=0}^{M} [\bigcup_{\{i:C_p(t_i)=0\}} a_{ij}]$$

where M is the total number of components in the system, and $[\cdot]$ is Iverson's operator, which casts logical True to 1, and False to 0.

In the example test matrix of Table I, only t_2 and t_5 are runtime testable without any preparation cost. On the other hand, t_1, t_3, and t_6 have $C_p(t_i) = 10$ since they cover c_7. t_4, t_7 and t_9 cover c_8 and hence have also $C_p(t_i) = 10$. Finally, since t_8 covers both c_7 and c_8, $C_p(t_8) = 20$.

The value of runtime testability measured by RTM is used to prioritize the expenses in preparation costs. This corresponds to a Binary Integer Programming problem, also known as the Knapsack problem. Since this is known to be NP-Hard different heuristics have been proposed [18].

III. Fault Diagnosis

The objective of fault diagnosis is to pinpoint the precise location of the fault (or faults) in the program by observing the program's behavior given a number of tests.

There is a large number of different diagnosis techniques (see Section VI). Our work is based on Spectrum-based fault localization (SFL), a well-known technique within Software Engineering. The main input of SFL is a coverage matrix, the same as the one in Table I. SFL also requires the binary outcomes, o_i (0: pass, 1: fail) of the test cases.

Automatic fault localization algorithms obtain a *ranking* of components by the likelihood of that component being at fault, for example by using statistical similarity coefficients [1], [23], or by using a Bayesian approach [2]. The ranking is returned to the user as the basis to find the faults. Typically the user finds the fault by inspecting each candidate in descending order according to the fault likelihoods.

A. Bayesian Diagnosis

In this paper we will use Bayesian diagnosis as the means to to obtain the diagnosis. The likelihood of a faulty component corresponds to the posterior probability of a component being faulty, given the outcome of the executed test, $\Pr(c_j|o_i)$, for a particular component c_j. We assume there can be only one faulty component.

Initially, for each component c_j, the probability of each explanation is $\Pr(c_j) = 1/M$, where M is the number of

components in the system. For each test case, the probability of each faulty component c_j is updated depending on the outcome o_i of the test, following Bayes' rule:

$$\Pr(c_j|o_i) = \frac{\Pr(o_i|c_j)}{\Pr(o_i)} \cdot \Pr(c_j) \quad (1)$$

In this equation, $\Pr(o_i|c_j)$ represents the probability of the observed outcome, if that diagnostic explanation d_k is the correct one, given by

$$\Pr(o_i = 1|c_j) = 1 - \Pr(o_i = 0|c_j) = a_{ij} \quad (2)$$

This formula is only valid if we assume test cases will produce no false negatives (i.e., tests that cover the fault but do not fail). For the purpose of our correlation study, this formula will hold.

$\Pr(o_i)$ represents the probability of the observed outcome, independently of which diagnostic explanation is the correct one. The value of $\Pr(o_i)$ is a normalizing factor that is given by

$$\Pr(o_i) = \sum_{d_k \in D} \Pr(o_i|d_k) \cdot \Pr(d_k) \quad (3)$$

and need not be computed directly.

B. Diagnostic Effort

We define C_d as the number of components the developer has to examine until finding the real fault c_* [2]. It corresponds to the position of c_* in the ranking. Because multiple components can be assigned the same probability, the value of C_d is averaged between the ranks of explanations that share the same probability, amongst which the real fault c_* is located.

$$C_d = \frac{|\{j : \Pr(c_j) > \Pr(c_*)\}| + |\{j : \Pr(c_j) \geq \Pr(c_*)\}| - 1}{2} \quad (4)$$

The above model for C_d is similar to existing diagnostic performance metrics [23], [27].

IV. Efficiency: Taking Testing Cost into Account

Each test that is executed provides new diagnostic information. As important as the final diagnostic effort, C_d, is the rate at which diagnostic effort decreases per unit of test cost. This is known as efficiency, S, and is calculated as

$$S = \sum_{n=1}^{N} (C_d(n) - C_d(N)) \quad (5)$$

where $C_d(n)$ represents the diagnostic effort after n test cases have been applied, and $C_d(N)$ is the diagnostic effort once all the test cases have been applied.

So far, the prioritization of test preparation costs has been driven exclusively by the trade-off between preparation cost and runtime testability/diagnosability. However, when considering fault localization, one should take into account not only the *final* residual effort of the diagnosis C_d, but also the testing cost involved. In other words, one should prioritize preparation operations that not only 'enable' tests that provide a good RTM and final C_d, but also that have a low S value.

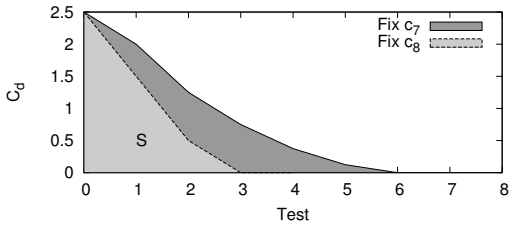

Fig. 1. Difference in S of two preparations with identical RTM

The following example matrix demonstrates how just RTM is not sufficient for this purpose.

$$A = \left[\begin{array}{cccccccc} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{array}\right]$$

Components c_7 and c_8 are not testable. If we prepare either of them, $RTM = 7$. However, if we take into account diagnostic efficiency, S, that each of the preparations yield, there is a clear difference. Figure 1 shows a simulation of how preparing c_7 will produce a much slower decay of C_d than if we had prepared c_8, even though the RTM values are identical for both options.

This example highlights the necessity of a third measurement that can predict the efficiency of the tests enabled by the preparations.

A. Runtime Diagnostic Efficiency

In the same way RTM allows us to prioritize the preparation costs of runtime tests according to their coverage, we study now a way to prioritize tests according to the efficiency, S, or the diagnostic they produce. To this ends, we propose the runtime diagnostic efficiency metric, RSM (for Runtime S Metric). We will use a simulation based on the evolution of the diagnostic effort when combined with a greedy optimization algorithm based on information gain [17]. The algorithm uses the tests that have $C_p(i_i) = 0$ to calculate the value of RSM, as was done with RTM.

Since simulating a diagnosis process can be very costly and requires many runs, we will use an approximation based on diagnosability. Every test matrix A decomposes a system into a partition of components $G = g_1, g_2, \ldots, g_L$, Each subset g_i contains all components with identical columns in A. Each of those subsets is called an *ambiguity group*.

The *diagnosability* of a test matrix is the ability of distinguishing between components uniquely, by decomposing the system into the smallest possible ambiguity groups. For

example the matrix

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

has a lower diagnostic ability than

$$A' = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

since the later generates a finer partition of ambiguity groups $\{\{1,3,6,9\},\{2\},\{5,8\},\{4\},\{7,10\}\}$ whereas the first one only $\{\{1,3,6,9\},\{2,5,8\},\{4,7,10\}\}$.

The probability of group g_i occurring is $\Pr(g_i) = \frac{|g_i|}{M}$, and the residual diagnostic effort in such group corresponds to $C_d = \frac{|g_i|-1}{2}$. The formula for the diagnosability is then

$$\mathrm{G}(A) = \frac{1}{M} \sum_{i=1}^{L} \frac{|g_i|}{M} \cdot \frac{|g_i|-1}{2} \tag{6}$$

and represents the expected (i.e., average) value of the diagnostic effort once the test in A have been applied.

The first example matrix in Table I has a $\mathrm{G}(A) = \frac{1}{10}(\frac{4}{10} \cdot \frac{3}{2} + \frac{3}{10} \cdot \frac{2}{2} + \frac{3}{10} \cdot \frac{2}{2}) = 0.12$. On the other hand the second example matrix, which can distinguish more components uniquely, has a $\mathrm{G}(A') = \frac{1}{10}(\frac{4}{10} \cdot \frac{3}{2} + 0 + \frac{2}{10} \cdot \frac{1}{2} + 0 + \frac{2}{10} \cdot \frac{1}{2}) = 0.08$.

1) Algorithm: This greedy simulation algorithm provides a good estimation of the evolution of C_d when using the information gain algorithm [17] to prioritize test cases, at a lower cost without having to resort to simulations of diagnoses.

Algorithm 1 RSM Estimation

$A \leftarrow \emptyset$
$T \leftarrow \{t_i : C_p(t_i) = 0\}$
$RSM \leftarrow \frac{M-1}{2}$
for $l \leftarrow 1, |T|$ **do**
  $i = \arg\min_{t_i \in T} (G(A||t_i))$
  $A \leftarrow A||t_i$
  $RSM \leftarrow RSM + G(A)$
return RSM

Algorithm 1 computes the evolution of $G(A)$ in a greedy way. At each step, it selects the test that minimizes the ambiguity of the matrix and appends the test to the matrix (operator $||$) . Initially the diagnosis is totally ambiguous ($C_d = \frac{M-1}{2}$). At every iteration, a test t_i is selected, such that, when concatenated to the matrix, it minimizes its ambiguity.

Obtaining the partition G has $O(M)$ time complexity when done incrementally. This has to be repeated N times per choice, for N choices. The complexity of Algorithm 1 is $O(MN^2)$, similar to IG but with a much lower constant, and it must be done once instead of being repeated a large number of times.

V. Experiments

Our evaluation study aims to answer the main research question concerning the utility of each of our metrics when considering the planning of preparation costs for runtime testing.

A. Systems

1) Example System I: WifiLounge: We diagnose the runtime testability and diagnosability of a wireless hotspot at an airport lounge [7]. The component architecture of the system is depicted in Figure 2. Clients authenticate themselves as either business class passengers, loyalty program members, or prepaid service clients.

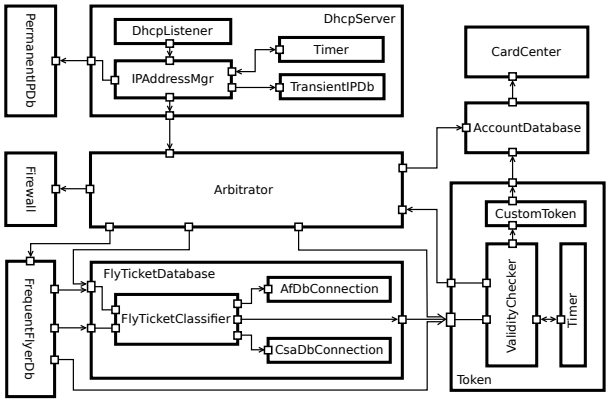

Fig. 2. Wifi Lounge Component Architecture

When a computer connects to the network, the `DhcpListener` component generates an event indicating the assigned IP address. All communications are blocked by the firewall until the authentication is validated. Passengers of business class are authenticated in the ticket databases of a number of airlines. Passengers from a frequent flier program are authenticated against the program's database, and the ticket databases to check that they are actually entitled to free access. Passengers using the prepaid method must create an account in the system, linked to a credit card that is used for the payments. Once the authentication has succeeded, the port block in the firewall is disabled so that the client can use the connection. The session ends when the user disconnects, or the authentication token becomes invalid. If the user is using a prepaid account, its remaining prepaid time will be updated.

2) Siemens & SIR Programs: Even though these programs and their test suites were not designed with runtime testing in mind, to strengthen the validity of our study, we use the test coverage matrices of a set of seven test programs known as the Siemens set [20], and five programs from the Software Infrastructure Repository (SIR) [8]. Table II provides more information about the programs (for more detailed information refer to [20] and [8]).

The coverage matrix A of each program is obtained by instrumenting each of the programs with `Zoltar` [21] to obtain the statements covered by each test case. Type and variable declarations and other static code, which are not instrumented by `Zoltar`, are excluded from diagnostic rankings and effort calculations.

Program	LOC	Tests	Description
print_tokens	563	4130	Lexical Analyzer
print_tokens2	509	4115	Lexical Analyzer
replace	563	5542	Pattern Matcher
schedule	412	2650	Priority Scheduler
schedule2	307	2710	Priority Scheduler
tcas	173	1608	Aircraft Control
tot_info	406	1052	Information Measure
sed	7125	370	Stream Editor
space	9126	150	ADL Compiler
grep	13287	809	String Matching
gzip	7933	210	Data Compression
flex	14194	107	Lexer Generator

TABLE II
SET OF PROGRAMS AND VERSIONS USED IN THE EXPERIMENTS

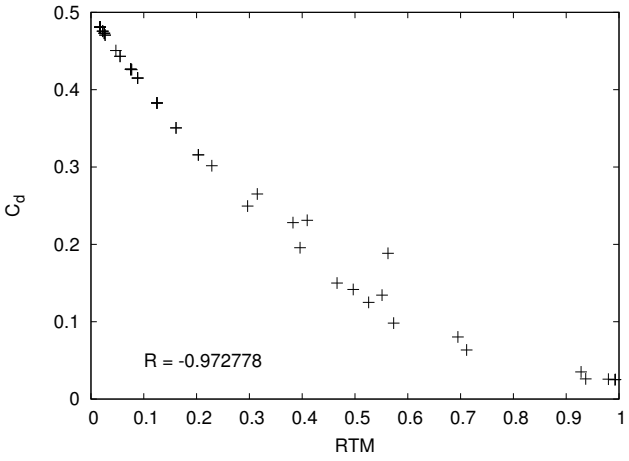

Fig. 3. Correlation between RTM and C_d

B. Relationship between RTM and C_d

Before going into considerations with test execution costs, the first question we want to answer is: what is the relationship between RTM and diagnostic effort, C_d, if any? To this end we randomly chose sets of 1 to 15 untestable components in each of the systems, and recorded the value of RTM, and C_d for a set of simulated faults. These faults were obtained by simulating random faults in one of the program components producing a failure for every test case that covers the faulty component.

Figure 3 shows the correlation between RTM and C_d. It can be clearly seen that there is a correlation between them, with a linear shape, or very slightly negative-exponential towards high values of RTM.

The reason for this direct relationship is that a testable (i.e., coverable) system, although not enough to guarantee a good C_d, is a prerequisite. By diagnosability, as explained in Section III, a good C_d is obtained by covering components in a way that makes them be in different ambiguity groups. The test matrices used in our study correspond to fairly diagnosable systems. Therefore, if a component is testable, it will be very likely also diagnosable, causing this direct relationship between RTM and C_d.

The main conclusion that can be drawn from this relationship is that an investment in runtime testability will cause a direct improvement on C_d. Since the relationship is very linear, we can be almost certain that our investment in testability has an equivalent return on the diagnosability of the system. The exception to this rule are systems with a large number of large ambiguity groups.

C. Relationship with Efficiency

A second question we would like to answer is the relationship with diagnostic efficiency. We have already presented an example in Section IV that illustrates the fact that a good runtime testability and diagnosability are no guarantee to an efficient diagnosis. This is clearly proven in the first of the two plots of Figure 4 that show that there is no correlation whatsoever between RTM, and diagnostic efficiency, S.

On the other hand, the RSM measurement, obtained by Algorithm 1 provides a much better correlation, and indicator of the efficiency that can be achieved. Moreover, RSM being based on $G(A)$ takes both into account the improvement rate of C_d, and the final C_d, making it a much more complete measurement.

D. Threats to Validity

The main threat to the validity of our experiments is the representativeness of the used systems. Adding a broader variety of systems where runtime testing is performed will enhance the validity of our results both because of the large sample of architectures but also because it would include more variety in the sources of test sensitivity.

A second threat stems from the fact that we used simulated faults in our validation. This could be improved by using mutation faults, since there is greater consensus that mutation faults are similar in nature to faults found during unit testing.

VI. RELATED WORK

Testability and testing effort were originally related to the probability of a fault causing an error, this error propagating to the output [19], [31], and the error being actually detected by the test oracle [3].

However, testability is affected by many other factors and its measurement has been studied extensively from a qualitative [4], [12] and quantitative [24], [11] way. Runtime testability and RTM were first introduced in [14] and refined in [18]. A method for the improvement of runtime testability of component based systems was described in [16]. In contrast with the previous work, in this paper we focus in the diagnostic perspective of runtime testing to optimize the runtime testing preparation costs.

Cost prioritization during testing has been extensively studied, from the point of view of test execution costs. Test prioritization techniques usually operate by selecting test cases that provided the highest fault detection potential[10], [22], [25], [28], [29], [32]. Lately, test prioritization which selects tests by their potential to locate faults has been proposed [17] potential. None of these works take into account preparation cost.

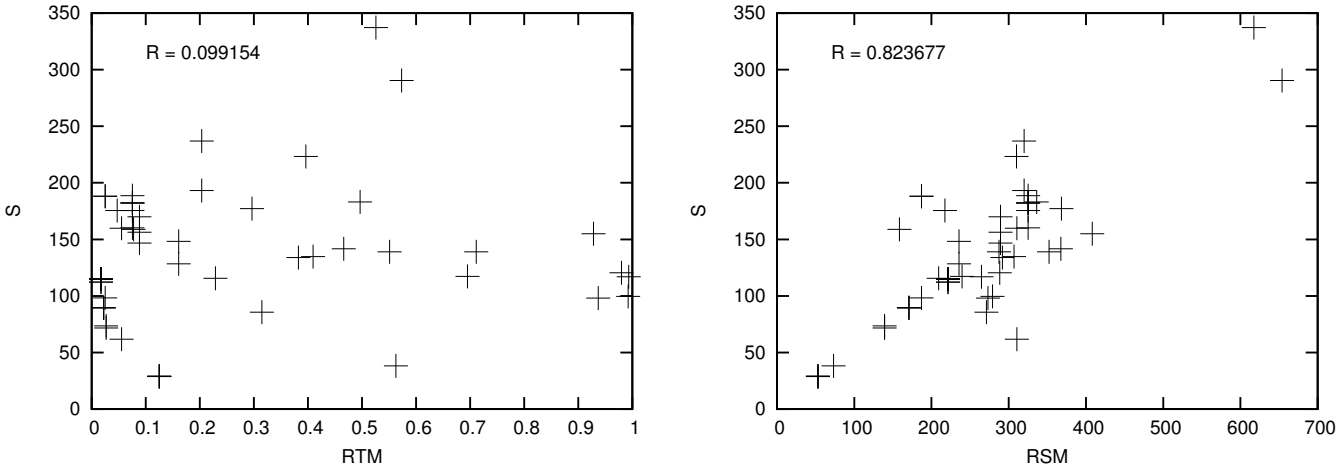

Fig. 4. Correlation of RTM, and RSM with Efficiency

VII. Conclusions & Future Work

Existing literature on runtime testability has considered only the implications of testability in the prioritization of preparation costs for runtime testing. In this paper we have studied runtime testing from the point of view of diagnosis. The main conclusions are (1) that can be drawn are that runtime testability and diagnosability are related strongly to each other, (2) that RTM and RDM have no relationship with the efficiency of the diagnosis. We have introduced a third measurement, RSM, based on a greedy algorithm that can predict the efficiency of runtime diagnosis with a good accuracy.

Future work includes expanding the set of systems used in our validation, and exploring less costly techniques for predicting runtime diagnosis efficiency than a greedy simulation algorithm.

Acknowledgements

This project is partially supported by the Dutch Ministry of Economic Affairs under the BSIK03021 program and the Poseidon project of the Embedded Systems Institute (ESI), Eindhoven, The Netherlands.

References

[1] R. Abreu, P. Zoeteweij, and A. van Gemund. On the accuracy of spectrum-based fault localization. In *Proc. TAIC PART'07*.

[2] R. Abreu, P. Zoeteweij, and A. van Gemund. Spectrum-based multiple fault localization. In *Proc. ASE'09*.

[3] 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.

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

[5] 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.

[6] 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.

[7] T. Bures. Fractal bpc demo.

[8] H. Do, S. G. Elbaum, and G. Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Emp. Soft. Eng. J.*, 10(4), 2005.

[9] 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.

[10] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE TSE*, 28(2), 2002.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] A. Gonzalez-Sanchez, E. Piel, H.-G. Gross, and A. van Gemund. Prioritizing tests for software fault localization. In *Proc. QSIC'10*.

[18] A. Gonzalez-Sanchez, É. Piel, H.-G. Gross, and A. J. van Gemund. Minimising the preparation cost of runtime testing based on testability metrics. In *34th IEEE Software and Applications Conference*, Seoul, South Korea, July 2010. (to appear).

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

[20] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments of the

effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proc. ICSE '94*.

[21] T. Janssen, R. Abreu, and A. J. C. van Gemund. ZOLTAR: A toolset for automatic fault localization. In *Proc. ASE'09 - Tool Demonstrations*.

[22] B. Jiang, Z. Zhang, W. Chan, and T. Tse. Adaptive random test case prioritization. In *Proc. ASE'09*.

[23] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proc. ICSE'02*.

[24] 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.

[25] Z. Li, M. Harman, and R. M. Hierons. Search algorithms for regression test case prioritization. *IEEE TSE*, 33(4), 2007.

[26] 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.

[27] M. Renieris and S. P. Reiss. Fault localization with nearest neighbor queries. In *Proc. ASE'03*.

[28] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. *IEEE TSE*, 27(10), 2001.

[29] A. M. Smith and G. M. Kapfhammer. An empirical study of incorporating cost into test suite reduction and prioritization. In *Proc. SAC'09*.

[30] 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.

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

[32] W. E. Wong, J. R. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In *Proc. ISSRE'97*.