Software Reliability Assessment based on Operational Representativeness and Interaction Coverage

Matthias Meitner, Francesca Saglietti
Chair of Software Engineering, University of Erlangen-Nuremberg
Erlangen, Germany

Summary

This article proposes a novel approach for quantitative software reliability assessment guaranteeing high component interaction coverage. For this purpose, measures for operational representativeness, independence of test cases and integration coverage are evaluated and optimized by multi-criterial genetic algorithms. They yield optimal test cases supporting the derivation of conservative reliability measures.

1 Introduction

Thanks to the benefits offered by re-using proven-in-use modules, component-based software systems are becoming increasingly attractive both in terms of economy and of quality. Even in case of newly developed modules, the component-based development paradigm helps enhancing design transparency and clarity of complex architectures. While easing the construction of complex design logic, this meanwhile widely diffused approach requires a particularly careful examination of component interactions. Numerous accidents that occurred during the last decades demonstrate the risks implied by faults at the interfaces of inherently correct components.

In the past, some approaches were devoted to the generation of integration test cases with respect to different interface coverage criteria [1, 5, 9, 12, 13, 15]. While thorough integration testing aiming at high interaction coverage may provide support in terms of fault detection, it does not address the question of a quantitative reliability assessment as required for safety-critical software systems. In fact, tests aiming at achieving coverage criteria in a white- or grey-box fashion are not related to any future operational profile and hence do not reflect an operating environment from which to derive reliability estimates.

On the other hand, there are sound and well-founded approaches supporting a conservative evaluation of software reliability via statistical sampling theory. Although the testing effort required to apply this theory may be considerable, the exploitation of operational experience as presented in [14] can help to enhance its practical applicability. Such approaches, however, are based on a black-box perspective, focused on the operational frequency of functional demands and fully neglecting the internal structure of software systems. For this reason, even allowing for the determination of confidence levels, these techniques cannot guarantee that the samples on which the reliability is estimated, actually represent the whole interaction spectrum offered by the application to be analyzed.

The approach proposed in this paper aims at the combination of both extremes by focusing on tests reflecting operational evidence and at the same time capable of achieving high interaction coverage. This allows for a quantitative software reliability evaluation based on operational evidence including an acceptable portion of the potential interaction space.

The paper is organized as follows:

- in ***section 2*** the basics of statistical sampling theory are summarized;
- ***section 3*** presents the objectives that the test case set has to fulfil in order to derive a reliability estimate and to guarantee a high degree of interaction coverage;
- in ***section 4*** a novel approach for combining these objectives is proposed, along with a short description of the optimization process;
- finally, ***section 5*** summarizes the contribution.

2 Reliability Estimation by Statistical Sampling Theory

Statistical sampling theory is a well-established approach to derive a reliability estimate for a software system [3, 10, 11]. For any given confidence level β and a large number n ($n > 100$) of correct test runs, an upper bound $\tilde{p}$ for the unknown failure probability p can be derived:

$$P(p \leq \tilde{p}) = \beta$$

The theory requires the following conditions to be fulfilled:

- ***Condition 1 - independent selection of test cases***
  the selection of a test case does not influence the selection of other test cases.

- ***Condition 2 - independent execution of test cases***
  the execution of a test case does not influence the outcome of other test cases.

- ***Condition 3 - operationally representative testing profile***
  the test cases are selected at the same probability as they are expected during operation.

- ***Condition 4 - any software failure would be noticed***
  obviously, missing software failures during testing would result in over-optimistic reliability estimates. The assumption of a dependable oracle usually relies on high expert judgment of the domain engineers involved in the verification procedure.

- ***Condition 5 - test success***
  no failure (or very few failures) occur(s) during the test. As the admittance of few failure observations [17] is achieved at the cost of lower reliability estimates, in the following we assume no failure occurrence, as considered appropriate for safety-critical applications.

In such a case, statistical sampling theory allows to derive the following relation between n, $\tilde{p}$ and β:

$$(1-\tilde{p})^n = 1-\beta$$

The number of test cases n that need to be successfully executed is:

$$n = \frac{\ln(1-\beta)}{\ln(1-\tilde{p})}$$

Table 1 shows the relation between the testing amount n and the failure probability upper bound $\tilde{p}$ at confidence level $\beta = 0.99$ for the 4 low demand SIL levels defined in IEC 61508 [4].

SIL	$\tilde{p}$	n
1	$\geq 10^{-2}$ to $<10^{-1}$	459
2	$\geq 10^{-3}$ to $<10^{-2}$	4603
3	$\geq 10^{-4}$ to $<10^{-3}$	46050
4	$\geq 10^{-5}$ to $<10^{-4}$	460515

Table 1 Relations between n and $\tilde{p}$ for $\beta = 0.99$

3 Multi-Objective Approach to the Generation of Optimal Test Cases

As the approach presented in this article focuses on the generation of adequate test cases, only conditions 1 and 3 (operationally representative and statistically independent test selection) will be considered in the following. Fulfilling conditions 4 and 5 may be delegated to the domain engineers and to the development resp. maintenance team, while condition 2 can be easily achieved by re-setting mechanisms.

Depending on the application considered, the input parameters can be semantically correlated, e.g.

- via ***physical laws***
  like electrical resistance, voltage and current, or

- via ***logical patterns***
  like relation of fields in an invertible matrix.

While dependencies determined by the physical or logical, application-specific environment are not modifiable, nor reducible, further correlations between test cases may arise by instantiation of the parameters, e.g. by numerical dependencies among parameter values.

For the purpose of reliability assessment, such dependencies have to be removed by filters based on correlation measures (s. section 3.2). An approach to quantify dependencies among non-numerical parameters like strings is proposed in [7].

Figure 1 summarizes both dependency layers mentioned above.

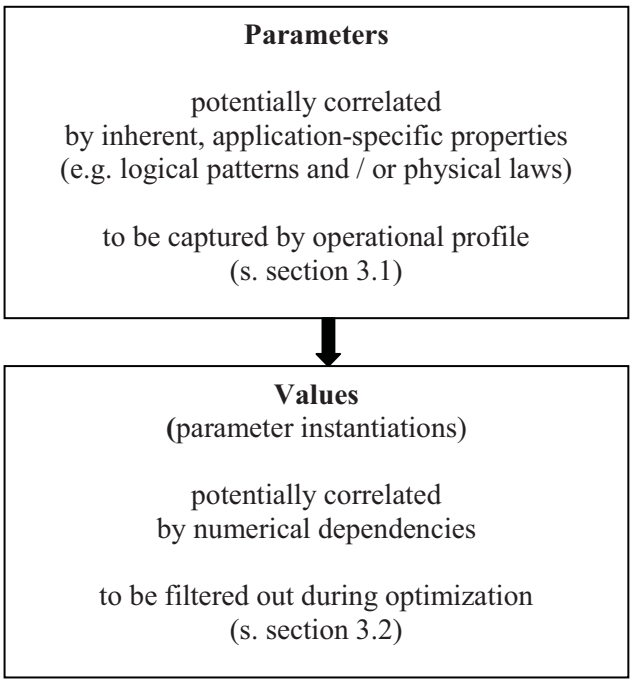

Figure 1 Dependencies among parameters and values

3.1 Objective 1: Operational Profile

We assume that the expected operational profile of the application under test has been preliminarily estimated on the basis of the frequency of occurrence of the input parameters.

As far as parameters can be considered independently, their values may be produced by random generators on the basis of a pre-defined distribution. On the other hand, parameters depending on others are deterministically instantiated via their functional dependence.

The validity of operational representativeness can be evaluated by goodness-of-fit-tests like the χ^2-test, checking whether the distribution of values observed is consistent with a pre-specified distribution (null hypothesis H_0). The value of the resulting χ^2-statistic is:

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

where k is the number of bins, O_i the observed and E_i the expected frequency for bin i ($1 \le i \le k$).

Hypothesis H_0 is rejected at a given significance level α if the value of the test statistic is greater than a critical value. If the value of the test statistic is lower than the critical value, the data may be taken as representative for the expected operational profile.

In addition to χ^2-test, further goodness-of-fit tests used in this work are the Kolmogorov-Smirnov-Test [6] and the Anderson-Darling-Test [6].

Figure 2 shows an example for the fitting of parameter values to a normal distribution.

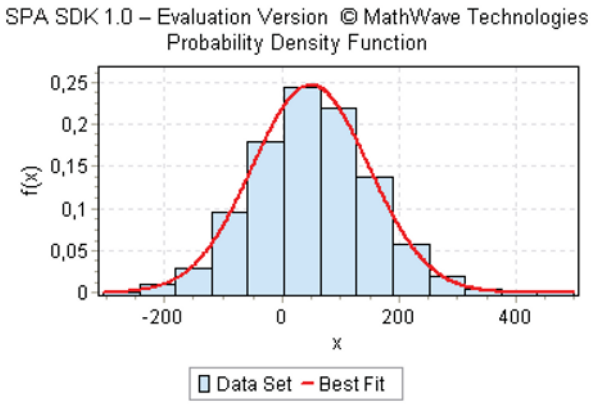

Figure 2 Distribution fitting

3.2 Objective 2: Test Case Independence

As mentioned above, statistical sampling theory assumes independently selected test cases. For the purpose of test case generation this means that values of parameters previously classified as independent must be uncorrelated. In order to evaluate potential correlations, both cross-correlation and auto-correlation measures are considered.

Cross-correlation addresses dependencies among different parameters within the same test case. One of the metrics for measuring cross-correlation used in this work is Pearson's product-moment correlation coefficient (PMCC) [2]:

$$r_{xy} = \frac{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)\cdot\left(y_i - \bar{y}\right)}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2}}$$

where n denotes the number of test cases and X and Y denote different random variables with values x_i resp. y_i ($1 \le i \le n$) and corresponding average values $\bar{x}$ and $\bar{y}$. PMCC measures the linear dependence of these two variables, with values close to 0 indicating that the variables may be considered as uncorrelated and values near 1 or -1 pointing to high positive or negative linear correlation. Further cross-correlation metrics used in this work include Spearman's rank correlation coefficient, Kendall's τ and Cramer's V measures [16].

On the other hand, auto-correlation describes the correlation among values of one parameter in different test cases. The term lag describes the distance between these test cases, e.g. auto-correlation with lag 1 measures the correlations of parameter values in successive test cases. Based on Pearson's correlation coefficient is the following measure for auto-correlation:

$$r_k = \frac{\frac{1}{n-k}\sum_{i=k+1}^{n}\left(x_i - \bar{x}\right)\cdot\left(x_{i-k} - \bar{x}\right)}{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}$$

where n denotes the number of test cases, k denotes the corresponding lag and X denotes a random variable with values x_i ($1 \le i \le n$) and corresponding average value $\bar{x}$.

In order to determine whether the values of the independent parameters of a test case set are uncorrelated, a maximum threshold has to be identified. For correlation test results below that threshold the parameter values are regarded as independent. Accordingly, correlation test results above the threshold denote relevant dependencies among parameter values.

3.3 Objective 3: Interaction Coverage

Classical approaches addressing structural testing coverage go back to the '60s [8]. While testing strategies based on control flow aim at covering the program elements reflecting potential transfer of control (like statements or

branches), testing techniques based on data flow extend these considerations to include the coverage of pairs consisting of variable definitions and of their usage(s).

While these criteria are useful for monolithic systems up to a certain size, they become too time-consuming when testing the integration of modules, in particular if the latter already revealed as proven-in-use. Therefore, new coverage criteria addressing component interfaces were defined, among them coupling-based criteria [5] examining the interaction between two program components, where one of the components (the *caller*) contains an invocation to the other component (the *callee*). The node in the control flow of the caller which contains the invocation of the callee is denoted a *call site*.

The testing criteria distinguish between the following coupling types:

- ***parameter coupling***
  refers to parameter passing;

- ***shared data coupling***
  refers to shared data objects.

A *coupling-def* denotes a node that contains a definition of a variable that can reach a use in another program component on some execution path. There are three types of coupling-defs:

- ***last-def-before-call***
  definition of a formal parameter before a call;

- ***last-def-before-return***
  definition of an actual parameter before a return;

- ***shared-data-def***
  definition of a shared variable.

A *coupling-use* is a node that contains a use of a variable that can be reached by a definition in another program component on at least one execution path.
Again, there are three types of coupling-uses:

- ***first-use-after-call***
  use of a formal parameter after a call;

- ***first-use-in-callee***
  use of an actual parameter inside a callee;

- ***shared-data-use***
  use of a shared variable.

A path is denoted a *def-clear path* with respect to a given variable if there is no definition of that variable along the path considered.

A *coupling path* between two program components is a path from a coupling-def to a coupling-use that is def-clear with respect to that variable.

Figure 3 shows an example for two coupling paths.

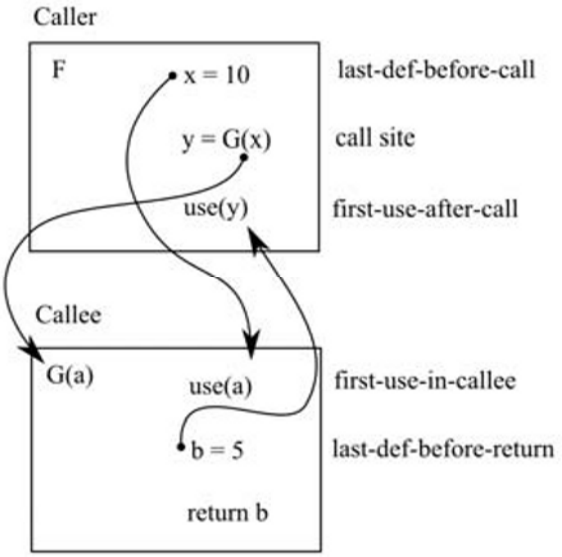

Figure 3 Examples for coupling paths

Using these concepts the following coupling-based testing criteria were defined in [5]:

- ***call coupling criterion***
  cover all call sites;

- ***all-coupling-defs***
  for each variable, cover at least one coupling path from each of its coupling-defs to one of its reachable coupling-uses;

- ***all-coupling-uses***
  for each variable, cover at least one coupling path from each of its coupling-defs to each of its reachable coupling-uses;

- ***all-coupling-paths***
  for each variable, cover all coupling paths sets (s. remark below) from each of its coupling-defs to each of its reachable coupling-uses.

These criteria can be arranged in the subsumption hierarchy shown in **Figure 4**.

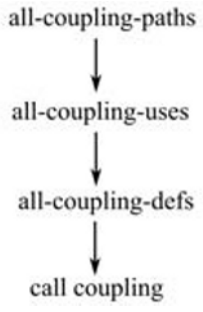

Figure 4 Subsumption hierarchy of coupling-based testing criteria (from [5])

As requiring the execution of all coupling paths may be impractical in general due to loops, test coverage may be restricted to so-called coupling path sets: a *coupling path set* is a set of nodes that can be reached on paths between a coupling-def and a coupling-use [5]. By covering all coupling path sets it is ensured that each loop body is executed at least once.

With the development of the object-oriented programming paradigm, complexity is moving even further from intra-modular functionality to component interaction. In particular, object-oriented mechanisms such as inheritance, polymorphism and dynamic binding can strongly affect component integration [1]. These faults are hard to detect and usually are not detectable by mere application of standard control and data flow testing techniques. Accordingly, new testing criteria were developed in [1] focusing on the interfaces of object-oriented code portions by considering three methods, where

- a so-called ***coupling method*** calls
- first a so-called ***antecedent method*** which defines variable(s),
- then a so-called ***consequent method*** which uses the same variable(s).

Both the calls to the antecedent and to the consequent method are carried out via the same instance variable (also called *context variable*). They build a so-called *coupling sequence.* To each coupling sequence there is an associated set of *coupling variables* defined in the antecedent method and used in the consequent method. A *coupling path* is a path between the definition and the use of a coupling variable, which is def-clear with respect to that variable.

Figure 5 illustrates the situation where o is the context variable and a coupling method f() calls an antecedent method m() and a consequent method n().

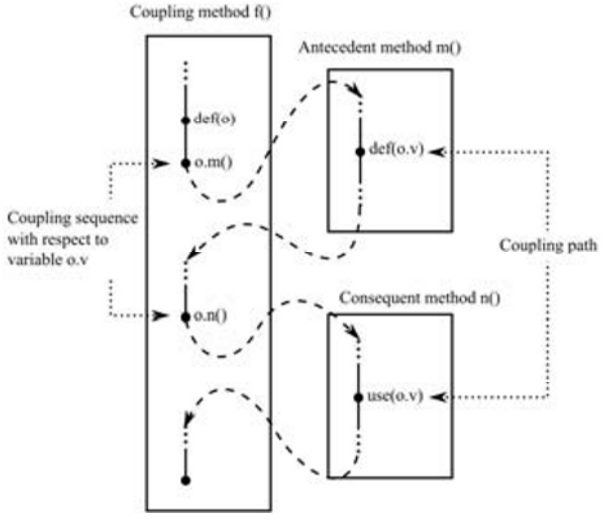

Figure 5 Coupling sequence and coupling path

Depending on the actual type of the context variable different polymorphic methods may be executed. Their variety is limited by the size of the type family of the context variable. Furthermore, the number of coupling variables defined and used can also vary because of polymorphism. Based on these definitions four criteria were proposed in [1] considering the effects of inheritance and polymorphism on the coupling sequences:

- ***all-coupling-sequences***
  for every coupling sequence, cover at least one coupling path induced by that coupling sequence;
- ***all-poly-classes***
  for every member of the type family of a context variable, cover at least one coupling path induced by the coupling sequence;
- ***all-coupling-defs-uses***
  for each coupling variable in a coupling sequence, cover all feasible coupling paths between its coupling-defs and its coupling-uses;
- ***all-poly-coupling-defs-uses***
  for every member of the type family of a context variable cover all feasible coupling paths between coupling-defs and coupling-uses of every coupling variable in a coupling sequence.

These criteria can be arranged in the subsumption hierarchy shown in **Figure 6**.

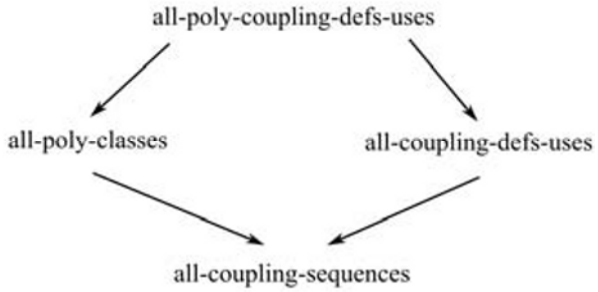

Figure 6 Subsumption Hierarchy of Object-Oriented Coupling-Based Testing Criteria (from [1])

4 Combination of Objectives

As already mentioned, the novel approach presented in this article aims at the combination of three different objectives:

- ***objective 1***
  operational representativeness (s. section 3.1);
- ***objective 2***
  test case independence (s. section 3.2);
- ***objective 3***
  interaction coverage (s. section 3.3).

As our major goal concerns a sound and well-founded software reliability estimation, the conditions underlying statistical sampling theory must be fulfilled. Therefore, the objectives 1 and 2 are "knock-out" criteria, in particular they dominate over objective 3; the latter one should be optimized without jeopardizing any of the former ones.

Because of the extremely high computational complexity involved in this problem, systematic techniques are inadequate and have to be replaced by heuristic approaches, in this case multi-objective optimization by genetic algorithms. They rely on the definition of an adequate fitness function assessing the individuals of successive populations by reflecting the degree at which they fulfil the objectives 1 - 3.

The optimization algorithm proceeds as follows:

- ***step 1***
  generate an initial population consisting of sets of test cases, each reflecting the operational profile;
- ***step 2***
  evaluate the fitness of each test case set by measuring the degree of fulfilment of the objectives 1 - 3;
- ***step 3***
  if a predefined stopping rule based on an acceptable fitness measure or on a maximum execution time is fulfilled, continue with step 5, otherwise with step 4;
- ***step 4***
  generate a new population by application of genetic operators (selection, cross over, mutation), followed by new iteration (go back to step 2);
- ***step 5***
  output the optimal test case set generated so far.

In order to capture the degree of fulfilment of objectives 1 and 2 in a quantitative manner, their degree of violation (evaluated via goodness-of-fit tests resp. correlation metrics) is mapped on the interval [0; 1], where

- **1** denotes full compliance with the criterion considered and
- **0** denotes inacceptable violation.

Denoting by t the maximum acceptable threshold for violation implies that

- all values x_1 between 0 and t (i.e. $x_1 \in [0; t]$) are assessed as fully compliant (thus rated as 1) while
- all values x_2 beyond t (i.e. $x_2 \in]t; max]$, where max denotes the maximum violation degree) cannot be taken as fully compliant and are rated below 0.9.

Figure 7 shows this normalization function for violation of representativeness, while **Figure 8** addresses correlation.

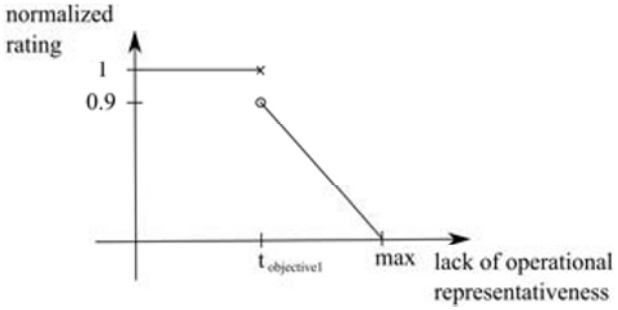

Figure 7 Normalization of lack of operational representativeness

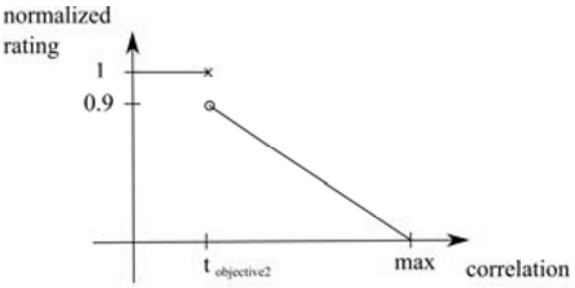

Figure 8 Normalization of correlation (max = 1)

On this basis the overall fitness function addressing all objectives 1 - 3 can be defined as the following weighted sum:

Fitness value =
0.1 · interface coverage achieved +
1.0 · rating for operational representativeness +
1.0 · rating for test case independence

The weights are selected such that even maximum interaction coverage (objective 3) cannot compensate the violation of at least one knock-out criterion (objectives 1-2). A test case set that does not violate any of the knock-out criteria has therefore a higher fitness value (≥ 2.0) than a test case set violating at least one of these criteria (< 2.0), no matter how high its interface coverage is.

Based on this fitness function the genetic algorithm can generate an optimal test case set supporting software reliability assessment as well as high interaction coverage.

5 Conclusion

This article proposes a new approach for software reliability assessment ensuring high interface coverage. The technique developed makes use of genetic optimization algorithms. For this purpose an adequate fitness function was defined, taking into account the 3 objectives of operational representativeness, independence of test case selection and high interaction coverage.

Acknowledgement

The authors gratefully acknowledge that the work presented was partly funded by Siemens Corporate Technology.

6 Literature

[1] Alexander, R. T., Offutt A. J.: Coupling-based Testing of O-O Programs, Journal of Universal Computer Science, vol. 10(4), 2004

[2] Hartung, J.: Statistik, Oldenbourg, 1995

[3] Ehrenberger, W.: Software-Verifikation, Hanser Verlag, 2002

[4] International Electrotechnical Commission, Functional Safety of Electrical / Electronic / Programmable Electronic Safety-Related Systems (IEC 61508)

[5] Jin, Z., Offutt A. J.: Coupling-based Criteria for Integration Testing, Software Testing, Verification & Reliability 8(3): 133-154, 1998

[6] Law, A. M., Kelton, W. D.: Simulation, Modeling and Analysis, McGraw-Hill, 2000

[7] Meitner, M.: Erfassung und Bewertung der mit Softwarekomponenten gewonnenen Betriebserfahrung, Diploma thesis, Chair of Software Engineering, University of Erlangen-Nuremberg, 2009

[8] Miller, J. C., Maloney, C. J.: Systematic mistake analysis of digital computer programs, Communications of the ACM, 6(2):58-63, 1963

[9] Oster, N., Saglietti, F.: Automatic Test Data Generation by Multi-Objective Optimisation, Computer Safety, Reliability and Security, Lecture Notes in Computer Science, Vol. LNCS 4166, Springer-Verlag, 2006

[10] Parnas, D., van Schouwen, J., Kwan, S.: Evaluation of Safety-critical Software, Communications of the ACM, 33(6), 1990

[11] Quirk, W. J. (ed.): Verification and Validation of Real-time Software, Springer-Verlag, 1985

[12] Rehman, M., Jabeen, F., Bertolino, A., Polini, A.: Software Component Integration Testing: A Survey, Journal of Software Testing, Verification, and Reliability (STVR), 2006

[13] Saglietti, F., Oster, N., Pinte, F.: Interface Coverage Criteria Supporting Model-Based Integration Testing, Workshop Proc. of 20th International Conference on Architecture of Computing Systems (ARCS 2007), VDE, 2007

[14] Söhnlein, S., Saglietti, F., Bitzer, F., Meitner, M., Baryschew, S.: Software Reliability Assessment based on the Evaluation of Operational Experience, Measurement, Modelling, and Evaluation of Computing Systems - Dependability and Fault Tolerance, Lecture Notes in Computer Science, Vol. LNCS 5987, Springer-Verlag, 2010

[15] Spillner, A.: Test criteria and coverage measures for software integration testing, Software Quality Journal 4: 275-286, 1995

[16] Storm, R.: Wahrscheinlichkeitsrechnung, mathematische Statistik und Qualitätskontrolle, Hanser Verlag, 2007

[17] Störmer, H.: Mathematische Theorie der Zuverlässigkeit, Oldenbourg, 1970