RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

International Workshop on the
Design of Dependable Critical Systems

ECOMODIS

Proceedings

**of the
International Workshop on the
Design of Dependable Critical Systems
"Hardware, Software, and Human Factors
in Dependable System Design"**

DDCS 2009

**September 15, 2009
Hamburg, Germany**

In the framework of
The 28th International Conference on
Computer Safety, Reliability and Security
SAFECOMP 2009

Edited by

Achim Wagner[1], Meike Jipp[1], Colin Atkinson[2] and Essameddin Badreddin[1]

[1] Automation Laboratory, Institute of Computer Engineering,
University of Heidelberg
[2] Chair of Software Engineering, University of Mannheim

ziti

International Workshop on
the Design of Dependable Critical Systems
September 15, 2009, Hamburg, Germany

Measuring the Dependability of Dynamic Systems using Test Sheets

Colin Atkinson, Florian Barth and Giovanni Falcone

Lehrstuhl für Softwaretechnik, Universität Mannheim,
68131 Mannheim, Germany
{atkinson, barth, falcone}@informatik.uni-mannheim.de

Abstract. Determining a system or component's dependability invariably involves some kind of statistical analysis of a large number of tests of its behavior under typical usage conditions, regardless of the particular collection of attributes chosen to measure dependability. The number of factors that can affect the final figure is therefore quite large, and includes such things as the ordering of system operation invocations, the test cases (i.e. the parameter values and expected outcomes), the acceptability of different operation invocation results and the cumulative effect of the results over different usage scenarios. Quoting a single dependability number is therefore of little value without a clear presentation of the accompanying factors that generated it. Today, however, there is no compact or unified approach for representing this information in a way that makes it possible to judge dynamic systems and components for their dependability for particular applications. To address this problem, in this paper we describe a new, compact approach for presenting the tests used to determine a dynamic system's dependability along with the statistical operations used to turn them into a single measure.

1 Introduction

Quantifying the dependability of software components and dynamic systems is a major challenge. In contrast with traditional "hard-wired" systems whose behavior remains fixed (or should remain fixed) as they execute, dynamic systems change their apparent behavior as time goes by – in other words, they remember the effects of previous operations and modify their behavior accordingly. According to this definition, most none trivial software systems and components are dynamic systems. Because of the memory effect, the dependability of dynamic systems cannot be calculated from a single metric derived by the repetitive application of a fixed evaluation criterion (e.g. MTTF from system failures or availability from system crashes etc.). Instead, the dependability of dynamic systems has to be determined from compound measures obtained by applying different evaluation criteria to the system's behavior using non-trivial scenarios resembling typical usage patterns. Only then does a dependability measure give a true estimate of the lilkihood that a dynamic system will deliver satisfactory service in a typical usage situation.

Intuitively, Dependability is a measure of the degree to which the users of a system can justifiably rely on the service it delivers – that is, its behavior. In general, there are numerous properties or attributes of a system that influence its dependability, including [1][2]:

- *Availability*: the readiness for usage
- *Reliability*: the continuity of correct service
- *Safety*: the non-occurrence of catastrophic consequences on the environment
- *Confidentiality*: the non-occurrence of unauthorized disclosure of information
- *Integrity*: the non-occurrence of improper alterations of information

However, combining these separate factors into a single dependability measure is a highly application specific problem and there is currently no widely accepted theory that can be applied in a general way across different domains. To address this problem, the Ecomodis project has developed an approach to dependability specification and measurement that uses user-defined acceptability functions to provide an application-specific measure of a service's acceptability [3]. By observing a systems behavior over a series of carefully defined tests that mimic its real usage environment a picture of the system's overall behavior can be developed, and by using hypothesis-measurement statistics from such fields as psychology, a measure of a system's likely dependability for new applications can be obtained.

This approach relies on the clear and precise description of the tests used to exercise a system as well as the system's response to those test. However, traditional testing technology provides no concise way of describing such complex test scenarios or how the results of the tests are combined into higher-order measures. The most common way of doing this today is to write a software program in a general purpose programming language like Java or C++ that performs all the tests and applies the necessary statistical calculations to the results. However, just as with mainstream testing techniques based on standard software packages such as JUnit [4], this approach has a number of drawbacks. First, the ingredients and approach used are only understandable to software engineers who are familiar with the programming language used. Domain experts and managers who are unfamiliar with programming are unable to understand such descriptions. Second, even for people with the necessary expertise, the important information is obscured in a lot of superfluous programming "scaffolding" needed to create correct programs in the language concerned. This not only obscures the key test information and makes it more difficult to see, it also makes the task of writing correct descriptions more arduous and error prone.

To address this problem, the Ecomodis project has developed a new test specification technique to support the Ecomodis dependability model [3]. This approach, known as "Test Sheets" [5], was developed to combine the simplicity and readability of tabular test definition approaches such as FIT [6] with the flexibility of programmatic test definition approaches such as JUnit into a single unified approach based on the ubiquitous metaphor of spreadsheets. As well as allowing simple sequences of operation invocations (test cases) to be defined with the same expressive power as programming languages (but without the superfluous programming scaffolding) the approach also allows test case definitions to be nested to arbitrary levels and parameterized in arbitrary ways. In contrast with programmatic approaches, test sheets can also describe the results of tests. To support the assessment of dependability, standard test sheets have been enhanced with (a) an additional set of columns which describe the satisfaction functions (in input test sheets) and the satisfaction values (in output test sheets) defined on operation input/output values, and (b) an additional row that allows statistical operations to be applied to these acceptability values and other values derived from the test. In this paper we provide an overview of this enhanced form of test sheet and explain the new features designed to support the measurement and specification of the dependability of dynamic systems. We first describe the basic principles behind test sheets and then show how they are used through a small case study.

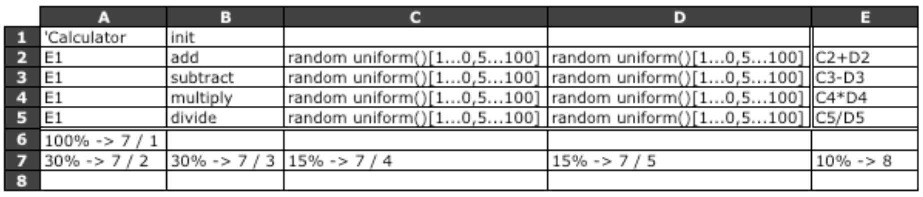
Expressing Usage Profiles with Test Sheets

The Test Sheet approach is a metaphor for test definition, application, and reporting which attempts to combine the power of programmatic approaches to testing with the readability and ease-of-use of tabular approaches. To achieve this goal a spreadsheet metaphor is used to identify the inputs to, and outputs from, operation invocations and express relationships between them. When complete, a language-independent test sheet can be transformed into source code in a specific target language for execution. Once executed, the test results can be visualized as a result test sheet. Furthermore, Test Sheets allow the definition of probabilistic or deterministic description of the test execution, thus allowing all kinds of behavioural protocols, algorithms [7] or any probabilistic operational profiles [8]to be defined.

To illustrate how test sheets support the process of measuring primitive dependability metrics and their combination into higher-order, compound metrics we use the example of a calculator. This component offers a number of mathematical operations that can be separated into two distinct groups that provide the basis for two different usage profiles:

- *basic operations*: add, subtract, multiply and divide
- *advanced operations*: log, sqrt, pow.

One usage profile characterizes applications that only use the basic operations of the calculator such as accounting applications. The other usage profile characterizes applications that also used the advanced operations such as scientific applications. Figure 1 and Figure 2 show the test definitions for the basic and the advanced usage profiles respectively.

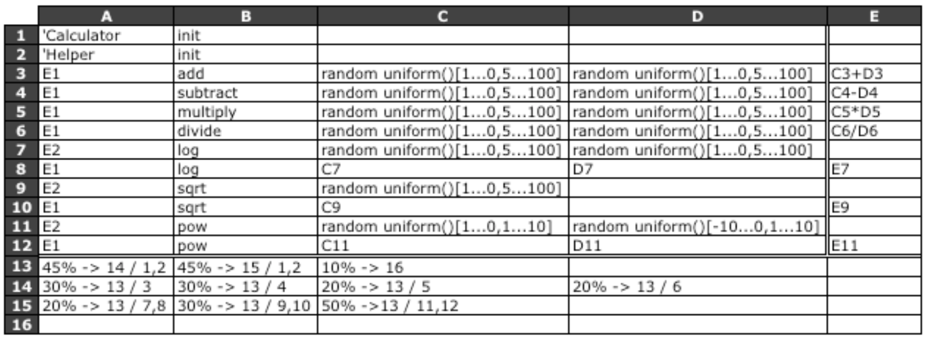


	A	B	C	D	E
1	'Calculator	init			
2	E1	add	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	C2+D2
3	E1	subtract	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	C3-D3
4	E1	multiply	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	C4*D4
5	E1	divide	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	C5/D5
6	100% -> 7 / 1				
7	30% -> 7 / 2	30% -> 7 / 3	15% -> 7 / 4	15% -> 7 / 5	10% -> 8
8					

Figure 1 Test Sheet for the basic usage scenario

The Test Sheet in Figure 1 tests the calculator component using a basic usage scenario. The first line initializes the component, while lines 2 to 5 invoke the basic arithmetic operations with random values. More specifically, line 2 invokes the add operation (cell B2) of the calculator object returned from first operation (cell A2) with two random values uniformly distributed between 1 and 100 with a step width of 0.5 (cells C2 and D2). The result of the computation is compared against the sum of the two parameters to determine its correctness (cell E2).

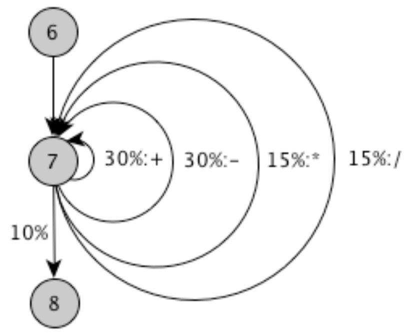
The order in which these operations are invoked is defined by the behavioral information in lines 6 to 8 which represents a simple state machine. Execution starts with line 6. Cell A5 states that with a probability of 100% the control flow will be transferred to line 7 after performing the operation invocation in line 1, the initialization. Cell A7 to E7 define the relative probabilities of subsequent operations. If any of the cells A7 to D7 is selected, the execution state returns to line 7 after the corresponding operation invocation is performed. However, if cell E7 is selected (which has a probability of 10%) the control flow will be transferred to line 8, which is empty, thus terminating the test execution. The state machine represented by this test sheet is shown as a UML state diagram in Figure 3.



	A	B	C	D	E
1	'Calculator	init			
2	'Helper	init			
3	E1	add	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	C3+D3
4	E1	subtract	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	C4-D4
5	E1	multiply	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	C5*D5
6	E1	divide	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	C6/D6
7	E2	log	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	
8	E1	log	C7	D7	E7
9	E2	sqrt	random uniform()[1...0,5...100]		
10	E1	sqrt	C9		E9
11	E2	pow	random uniform()[1...0,1...10]	random uniform()[-10...0,1...10]	
12	E1	pow	C11	D11	E11
13	45% -> 14 / 1,2	45% -> 15 / 1,2	10% -> 16		
14	30% -> 13 / 3	30% -> 13 / 4	20% -> 13 / 5	20% -> 13 / 6	
15	20% -> 13 / 7,8	30% -> 13 / 9,10	50% ->13 / 11,12		
16					

Figure 2 Test Sheet for the advanced usage scenario

Figure 2 shows the test sheet representing the advanced usage scenario. In order to validate the return values of the advanced operations, a helper component is introduced that serves as a test oracle. The helper object is initialized in line 2 after the initialization of the calculator component. Lines 7 through 12 show how results returned by invocation of the advanced operations (lines 8, 10 and 12) are verified using results derived from operations of the helper component (lines 7, 9 and 11 respectively). In line 7, the log operation of the helper is invoked to obtain the value used to verify the result returned by the calculator's log operation. In line 8, that log operation is invoked just like the basic operations in lines 3 to 6. However, in this case the input parameters are exactly the same as those used in the invocation of the helper component, indicated by the references to cells C7 and D7. The result of the calculator's log operation can thus be verified by comparing the value returned by the calculator to that returned by the helper component (cell E8). As before, the order of operation invocations is determined by the behavioural part of the test sheet in lines 13 through to 16. This is illustrated as a state diagram in Figure 4. In line 13 a decision is made whether to execute either a basic operation (line 14), an advanced operation (line 15) or to terminate the test execution (line 16) with the specified relative probabilities. In lines 14 and 15, the lines that represent the operations of the calculator are executed and the control flow is transferred to line 13 again, thus starting another loop through the algorithm.



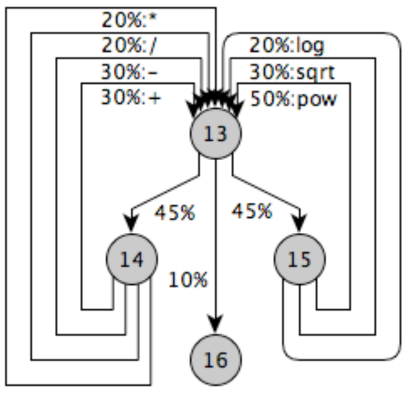


Figure 3 State diagram basic scenario

Figure 4 State diagram advanced scenario

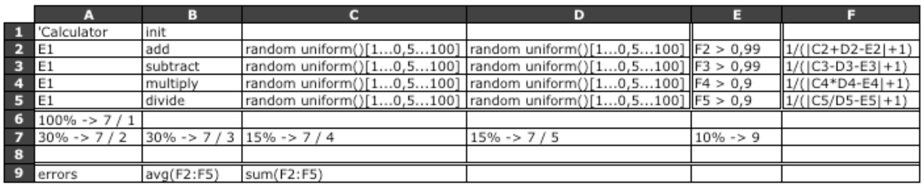
Test Sheet Extension for Dependability Measurement

The test sheets shows in the previous section are "standard" test sheets that can be used to define ordinary tests. Their strength is that by supporting the definition of behavioral information, components can be tested using realisitic, non-trivial scenarios. This provides the basis for obtaining meaningful dependability measures. However, it does not yet support the application of acceptability functions, nor the combination of acceptability values into higher-order measures. To support these, two further enhancements are introduced: acceptability cells and summary cells.

Acceptability Cells

To support the application of acceptability functions, an additional column group called *acceptability cells* is added to the right side of the standard test sheet layout (see Figure 5 Figure 7). These columns allow one or more satisfaction values to be calculated as defined by the equation or operation invocation in each cell. The contents of these cells can use the full expressive power of test sheets, like arithmetic expressions, cell references, etc. This allows the computation of complex metrics based on the runtime behavior of the component being tested.

Figure 5 shows the enhanced tests sheet corresponding to the simple scenario in Figure 1, illustrating the use of acceptability cells. The test sheets measures the acceptability of the results returned by the basic operations by computing the absolute delta between the returned and expected values. Line 2, in Figure 5 invokes the add method of the calculator component and the resulting value is stored in cell E2.



	A	B	C	D	E	F
1	'Calculator	init				
2	E1	add	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	F2 > 0,99	1/(|C2+D2-E2|+1)
3	E1	subtract	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	F3 > 0,99	1/(|C3-D3-E3|+1)
4	E1	multiply	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	F4 > 0,9	1/(|C4*D4-E4|+1)
5	E1	divide	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	F5 > 0,9	1/(|C5/D5-E5|+1)
6	100% -> 7 / 1					
7	30% -> 7 / 2	30% -> 7 / 3	15% -> 7 / 4	15% -> 7 / 5	10% -> 9	
8						
9	errors	avg(F2:F5)	sum(F2:F5)			

Figure 5 Enhanced Test Sheet for Basic Usage Scenario

The formula for the acceptability value first computes the delta between the returned and the expected result:

$$C2+D2 – E2$$

and then computes the absolute value of that delta:

$$|C2+D2 – E2|$$

This absolute delta is then put into a normalization function, in this case:

$$1/(x+1)$$

Figure 6 shows a plot of this function. The domain of the function is [0,∞] while the codomain is (0,1]. Hence the maximum value of the function is 1 for x = 0. The function is monotonically non-increasing so the value decreases for x-values greater then 0.

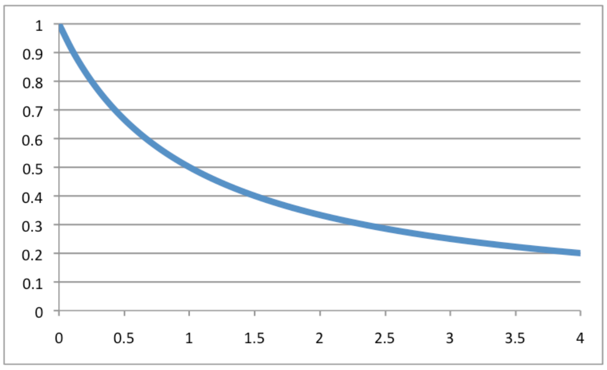


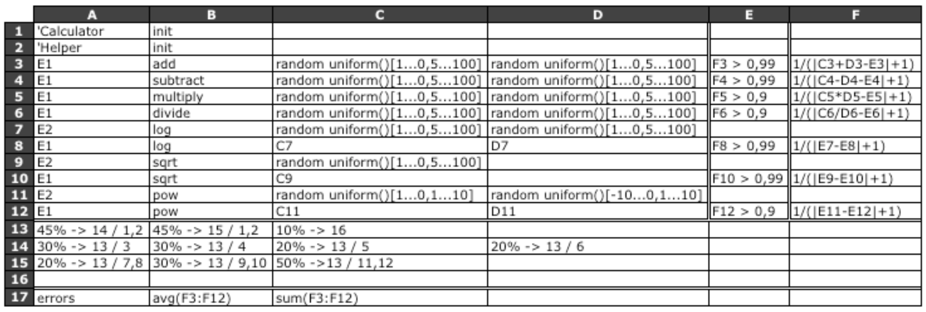
Figure 6 Normalisation Function: 1 / (x+1)

The complete formula is:

1/(|C2+D2-E2|+1)

Notice that this is only one possible acceptability function and the user is free to define computation formulas as needed. An additional option that becomes possible with acceptability cells is to use them to define the binary fail/pass criterion in the associated "Result Cells". In this case, the test will be marked as failed if the value of the satisfaction function is lower than 0.99 (cell E2).

Figure 7 shows the enhanced test sheet for the advanced usage scenario. In this case the acceptability of the advanced operation is calculated with a helper component. Instead of calculating the expected result inline as with the basic operations, the result of the helper component and the returned result of the calculator component are directly plugged into the function described before.



	A	B	C	D	E	F
1	'Calculator	init				
2	'Helper	init				
3	E1	add	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	F3 > 0,99	1/(|C3+D3-E3|+1)
4	E1	subtract	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	F4 > 0,99	1/(|C4-D4-E4|+1)
5	E1	multiply	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	F5 > 0,9	1/(|C5*D5-E5|+1)
6	E1	divide	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]	F6 > 0,9	1/(|C6/D6-E6|+1)
7	E2	log	random uniform()[1...0,5...100]	random uniform()[1...0,5...100]		
8	E1	log	C7	D7	F8 > 0,99	1/(|E7-E8|+1)
9	E2	sqrt	random uniform()[1...0,5...100]			
10	E1	sqrt	C9		F10 > 0,99	1/(|E9-E10|+1)
11	E2	pow	random uniform()[1...0,1...10]	random uniform()[-10...0,1...10]		
12	E1	pow	C11	D11	F12 > 0,9	1/(|E11-E12|+1)
13	45% -> 14 / 1,2	45% -> 15 / 1,2	10% -> 16			
14	30% -> 13 / 3	30% -> 13 / 4	20% -> 13 / 5	20% -> 13 / 6		
15	20% -> 13 / 7,8	30% -> 13 / 9,10	50% ->13 / 11,12			
16						
17	errors	avg(F3:F12)	sum(F3:F12)			

Figure 7 Enhanced Test Sheet for Advanced Usage Scenario

Summary Cells

To allow high-order values to be derived from the information in the acceptability and result cells a new area containing the *summary cells* has been introduced beneath the definition of the behavior. These cells not only contain the formulas or invocations used to determine new higher-order values, they also represent return values of the test sheet for potential use in higher order test sheets. The keyword "errors" generates a list of cells that failed the check against the expected result. Similar to the formula notation supported by spreadsheets users may define arbitrary formulas for the calculation of further return values.

Summary cells enable the user to define computations that summarize the behaviour of the component during the test, thus allowing the definition of dependability metrics and the calculation of higher order measures in a consistent and readable way. Using higher-order test sheets that allow test sheets to be arranged in hierarchies, the return values of the test sheet invocations, i.e. the lower level summary measures, can be used for further computations. This allows different dependability measures to be further merged into a single compound measure. The test sheets in Figures 5 and 7 contain summary cells. In the case of the basic usage scenario (Figure 5) the first summary value is the list of failed cells, the second is the average normalized deviation from the reference result, and the third is the normalized deviation. In the case of the advanced usage scenario (Figure 7) the same summary values are computed. Notice that in this case only cells that carry a value are relevant to the calculation, e.g. cell E7 will be left out when calculating the average or sum.

Conclusion

In this paper we described how test sheet can be used to support the measurement and specification of system dependability, and presented two enhancements to standard test sheets introduced for this purpose. Because of tests sheets' ability to define behavioural information it is possible to test dynamic systems with realistic usage patterns, thus enabling the assessment of meaningful dependability measures. The first enhancement to standard test sheets is the introduction of a new column group to support the application of acceptability functions. These complement the result cells (that represent a binary decision on a test's success) using a continuous measure for the evaluation of the test results. The second enhancement is the introduction of a row group for the application of statistical operations to the test's return values. This facilitates the computation of compound measures and their presentation in a consistent and understandable way.

References

[1] Melhart, B.; White, S., "Issues in defining, analyzing, refining, and specifying system dependability requirements," Engineering of Computer Based Systems, 2000. (ECBS 2000) IEEE Proceedings.
[2] Randell, B., "Dependability-a unifying concept," Computer Security, Dependability and Assurance: From Needs to Solutions, 1998. Proceedings , vol., no., pp.16-25, 1998
[3] C. Atkinson, A. Wagner and E. Badreddin, Towards a Practical, Unified Dependability Measure for Dynamic Systems, Workshop on the Design of Dependable Critical Systems, Hamburg, 2009.
[4] K.Beck. Test Driven Development: By Example, 2002.
[5] C. Atkinson, D. Brenner, G. Falcone, M. Juhasz. Specifying High-Assurance Services. IEEE Computer, vol. 41, no. 8, pp. 64-71, 2008.
[6] R. Mugridge and W. Cunningham, FIT for Developing Software. Framework for Integrated Tests, Robert C. Martin, 2005.
[7] H. Bär. Statische Verifikation von Softwareprotokollen. PhD thesis, University Fridericiana of Karlsruhe, 2004.
[8] J. D. Musa. Operational Profiles in Software-ReliabilityEngineering. IEEE Software. 10, 2 (Mar. 1993), 14-32, 1993.