

The Probabilistic Correctness of Conformance and Interoperability Testing

The Probabilistic Conformance & Interoperability Correctness Theorem

Rik Drummond

Drummond Group Inc -- GridWise Architecture Council
Fort Worth, Texas 76132 USA

rikd@drummondgroup.com

Keywords: Conformance, Interoperability, Testing, Correctness, Probability

Abstract

The common understanding of interoperability and conformance testing and their interrelatedness is fraught with bad assumptions and false ideas. Myths like conformance tested products are automatically interoperable and interoperability tested products are automatically conformant lead to greatly diminished returns within eBusiness systems and supply chains. Testing programs intended to help a community can instead hinder it if wrong conclusions are made regarding the interoperability and conformance of its products. Yet, without a widely accepted method and understanding of interoperability and conformance testing, the cycle of unmet expectations and undelivered promises will continue. There is a great need for a universal method to analyze and predict real-world interoperability and conformance of different testing processes.

This paper provides logical proofs and mathematic theorems to provide this needed analysis. The paper works out the mathematical basis for the probability of conformance and interoperability of testing procedure. Understanding and application of this probability analysis allows for implementers to better assess the expected results from certified testing programs.

This paper provides a logical and mathematical foundation for guidance in answering critical questions a test program must consider, such as:

- How many implementations must be tested for an interoperable product or a conformance engine to become reasonably conformant?
- How do you test for both interoperability and conformance?

- Why are eBusiness implementations problematic in testing for achieving both interoperability and conformance?
- Why do we have to be careful if organizations developing the products and the conformance or interoperability testing organization have significant communication about the standard?
- Do we always need to test for both conformance and interoperability or are their cases where we can save resources by only doing one and achieve or closely achieve the other?

1. INTEROPERABILITY AND CONFORMANCE TEST STRUCTURES

1.1. Introduction

In order to purposefully discuss interoperability (IOP) and conformance testing, it is important to fully comprehend the industry concepts and lingo in this arena. Several concepts must be discussed to enable a clear understanding of the various complexities and nuances involved in both types of test structures.

These tests are verifying the accuracy of various implementations of a specification. A *specification* is a pre-test agreement among implementers with sufficient detail and exactness as to allow the evaluation of an individual implementation's accuracy with regard to meeting the specification's conditions. This covers profiling as well as specifications that are not standards but are done when two or more companies decide to intercommunicate in a more ad hoc manner. Finally, it *may* cover all standards, such as HTTP, on which the specification is based. This is necessary because a specification is often tested for conformance or interoperability (IOP), yet does not test the supporting standards. These base standards may not be conformant in the implementations and could potentially cause an interoperability problem. For example, when using HTTP,

the systems under test (SUTS) do not know whether the code is conformant to HTTP specifications or if it has been profiled correctly across all the implementations.

1.2. Conformance Test

A conformance test of an application shows that the application conforms to the specification by interacting with the **conformance engine application**. During this type of test, the conformance engine (CE) generates output and receives input which is evaluated by implementations R1 through Rn-1 (Figure 1) and the CE. Both input and output from the interaction with the CE are expected to be conformant to the specification. The CE output is NOT evaluated by the conformance engine itself, because it is expected to be correct. The only verification that the CE output is correct comes by consensus from the participating systems R1 through Rn.

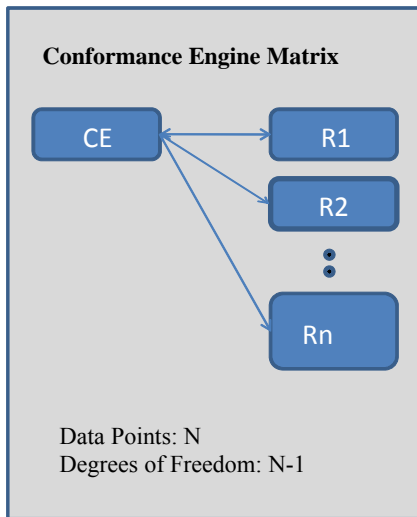


Figure 1

An implementation of a specification is said to be **conformant to the specification**, IF and only IF, the input $domain(x, y, \dots)$, and the output $range(a, b, \dots)$ of the implementation meet the requirements of the specification and the relation, $range = \mathbf{R}(domain)$, when \mathbf{R} implements the requirements of the specification. This is a normal Black-box with input and output. \mathbf{R} is the BLACKBOX, the input being the domain and the output being the range. See Figure 2. The dependant and independent variables of the range and domain may be Boolean, real, integer, documents, sets, etc. Therefore, the variables may be composed of any length bit-stream.



\mathbf{R} is a mathematical relationship. \mathbf{R} acts like a Black Box for testing purposes.

Figure 2

1.3. Full Matrix Interoperability Test

A full matrix interoperability test (Figure 3) for a set of

Interoperability (IOP) Test Structure

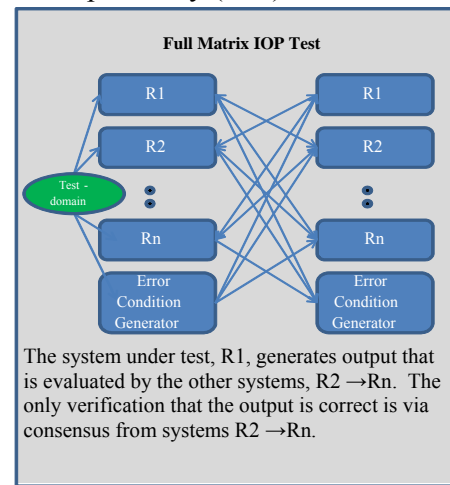


Figure 3

applications built on a **peer-interoperable specification** shows the applications interact properly – are peer-interoperable. Each system must initiate and respond with every other implementation in a full matrix manner as the *specification* states. Thus it must show that $\mathbf{R2}$ initiates and $\mathbf{R1}$ responds, $\mathbf{R1}(\mathbf{R2}(domain))$, and $\mathbf{R1}$ initiates and $\mathbf{R2}$ responds, $\mathbf{R2}(\mathbf{R1}(domain))$, properly. This is a composite relation. Also, both are a subset of *domain* for every pair of applications, whose relations $\mathbf{R1}$ through \mathbf{Rn} are within the test and an application responds to peer implementation $\mathbf{R}(\mathbf{R}(domain))$. See Figure 4.

A *specification* is said to be peer-interoperable, IF and only IF, the input $domain(x, y, \dots)$ and the output $range(a, b, \dots)$ meet the requirements of the *specification*. It also requires that the relation, \mathbf{R} , implements the requirements of the *specification* and *domain* is a superset (\supseteq) or proper superset (\supset) of $\mathbf{R}(\mathbf{R}(domain))$. (Figure 4) It is important to remember the dependant and independent variables of the

range and domain may be Boolean, real, integer, documents, sets, etc. Once again, the variables may be composed of any length bit-stream.

1.4. Relation - R

In addition to comprehending the models of conformance and interoperability testing, it is important to understand the mathematical concept of a relation. Suppose **R** is a relation from A to B. Then **R** is a set of ordered pairs where each first element comes from A and each second element comes from B. That is, for each pair $a \in A$ and $b \in B$ then $(a,b) \in R$ is read as “a is **R**-related to b”. The domain of a relation **R** is the set of all first elements of the ordered pairs which belong to **R**. The range of **R** is the set of second elements [1]. Each variable, a and b from (a,b) could each represent a set of (l, m, n,...).

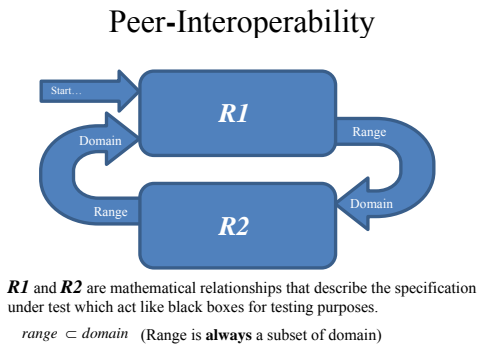


Figure 4

A relation, unlike a function, may have more than one correct output for exactly the same input. Thus a relation could have something such as (a,b) and (a,d), both being a correct response for an input of ‘a’. The relationship, versus the function, was selected for this series of definitions to make the definitions as general as possible.

In the interoperability definition, the idea of a **composite relation** ($R1 \circ R2$) is revealed. Let **R1** be a relation from A to B and let **R2** be a relation from B to A.(Figure 4) Then $(a, a) \in R1 \circ R2$ where $(a, b) \in R1$ and $(a, b) \in R2$. In a **peer- interoperable specification**, **R1** and **R2** are different representations of the same specification.

2. THEOREM: THE PROBABILISTIC CONFORMANCE & INTEROPERABILITY CORRECTNESS THEOREM

2.1. Theorem:

Any individual implementation of a set of size N implementations of **peer-interoperable specifications**, which are peer-interoperable among themselves, has the same probability of being conformant as a **conformance engine of error degree N-1**, if:

- the implementations are developed in a manner that produces random errors.
- the appropriate **error generator application** is part of the interoperability test.
- the same **test criteria** is used for both.
- the conformance engine was tested against itself.

2.2. Corollary:

Based upon test criteria, a Conformance Engine (CE) tested against N implementations or any single implementation tested against N other implementations in a full matrix IOP test has a probability of being conformant to the specification of:

$$(1 - APE^N)^M$$

Where N=number of implementations,
M=number of test cases,
APE = average probability of a **test-error** in an implementation on a test case.

NOTE: The Theorem and the Corollary will be proved concurrently below.

2.3. Pre-Proof Discussion:

In Figure 5, both methods have a flaw in that each ‘may’ not identify some *test-errors* or test-discrepancies. In statistics, these standard errors are generally referred to as Type I & Type II errors.^[2]

Type I Error: Rejecting a true null hypothesis. This can be restated as:

- Rejecting a true IOP system or a truly conformant system
- Reporting a test-error when the systems are truly conformant to the specification

Type II Error: Failing to reject a false null hypothesis (**testing event error**)

- Failing to identify a non-conformance error
- The systems under test agree that something is not a *test-error* when it actually is
- An error escaping thru the test regime for each test case

Throughout this proof, the discussion centers on type II errors.

The Key Question from which the theorem and corollary are produced is:

How can systems be interoperable based on a specification and not be conformant to that specification?

If one keeps this question in mind the proof will be easier to understand.

This situation can happen when all of the implementations in an IOP or conformance test make exactly the same ‘non-conformant error’. Each system would report a condition as ‘not an error’ when tests are conducted against the conformance engine or among each other. Henceforth, this situation involving a type II error will be referred to as a **testing event error**.

In Figure 5, the IOP test R1 would have to see the same test-error as a non-error for the N-1 other products in the test. In the **conformance test**, the

Theorem 3 View of Testing

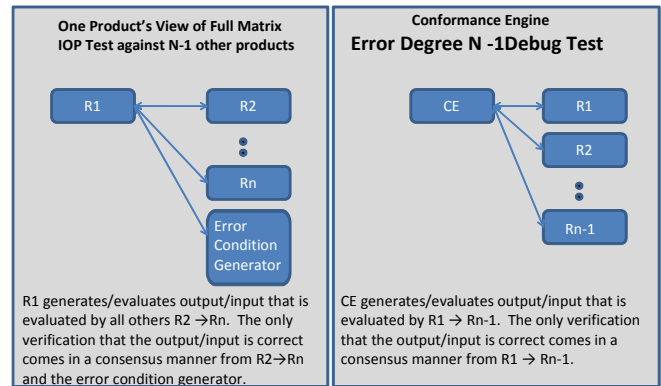


Figure 5

conformance engine would have to see the same non-error for the N products in the test in order for a real error to escape the test. The only way for this to happen is if R1 thru RN agree that a real test-error is not an error. If one implementation found the *test-error*, the specification should be checked to see if it is an error and correct all systems R1 thru RN as necessary. The same argument follows below for both test types: interoperability and conformance.

2.3.1. Example:

The specification requires that when an implementation receives an ‘A’ it should then respond with a ‘B’. This is the correct conformant response. However, if all of the implementations think that when they receive an ‘A’ they should respond with a ‘C’, then the systems all deem this as a correct response. In this situation all of the implementations pass the test. The conformance engine even passes because it is wrong also. Therefore, since all participants passed the test, then they all work. Yet, they are not conformant! The only way this can happen is if they all think sending the ‘C’ is correct. If even one of them thinks ‘C’ is incorrect, then a discrepancy will have been identified. With the proper research, the problem can be corrected.

This can be stated mathematically as:

Given any $a \in \text{domain}$, $b \in \text{range}$, and $c \in \text{range}$, the expected relation is:

$$(a, b) \in R1, (a, b) \in R2, \dots, (a, b) \in Rn \text{ and } (a, b) \in CE$$

However in a **test event error** situation, the actual result is

$$(a, c) \in R1, (a, c) \in R2, \dots, (a, c) \in Rn \text{ and } (a, c) \in CE$$

In this case, the test domain element (a) produces the same non-range **test-error** in all implementations – (a, c) where c is not part of the range as specified by the specification or it is the improper range element for this input ‘ a ’. This includes both the implementations and the CE -- conformance engine. Now, assume the CE sets a baseline validation against the same set of **RI** through **Rn** implementations which all contain the same **test-error**. The CE could cause problems in future conformance tests by confirming that the non-range or **test-error** element is correct when it is actually incorrect for a specific domain element. A full matrix IOP test on that same set of implementations will also encounter this exact issue. If the conformance engine is completely accurate in its implementation of the *specification*, the above scenario cannot happen. However, the only way to verify that it is completely true in its implementation is to test the CE against a number of implementations or other CE’s. These systems may be colluding together to cause an incorrect output from the CE. The goal of this contrived agreement is to get the CE to return a flag of correct when it is should be returning a flag of error. The collusion would happen more often if the applications (SUTS) were developed as a group effort. This conspiracy among the implementations usually only happens when the errors in the implementations are not random in nature.

2.4. Proof:

Theorem: Any individual implementation of a set of size N implementations of peer-interoperable specifications, which are peer-interoperable among themselves, has the same probability of being conformant as a conformance engine of error degree N-1, if: the implementations are developed in a manner that produces random errors.

- the appropriate **error generator application** is part of the interoperability test.
- the same **test criteria** is used for both.
- the conformance engine was tested against itself.

The test structure is looking for any conflicts between the implementations during *peer-interoperable* full matrix testing or verification of the conformance engine. When conflicts are discovered, they are resolved by referencing the *specification*. Thus, the issue is in identifying any conflict and then resolving the coding errors or interpretation issues in all implementations in a manner that meets the *specification*.

As tests against the N systems which have supposedly been programmed to the specification are conducted, what is the probability of a testing conflict NOT being reported when there really is a test-error? The implementation passes the test yet remains non-conformant.

Once a conflict is identified, verification must occur that the conflict ‘does or does not’ meet the specification. A conflict that is not revealed during a peer-interoperability test or validation of the CE on all N implementations is a test-event-error. The probability that **ALL** implementations make exactly the same non-conformant error in their implementations on one test case is:

$$(APE)^N$$

where APE is the average probability of a specific test error in an implementation.

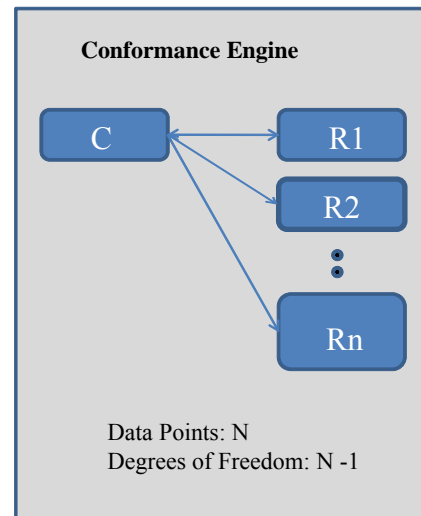


Figure 6

So again, $(APE)^N$ is the probability of an error escaping thru the test regime for each test case. The expression below represents the chances of identifying **test-errors** for

all test participants for one test case because one to several of the implementations identify this as an error. That is, zero errors are escaping through the test for each test case.

$$1 - (APE)^N$$

What is the probability of a situation like this happening for ALL test cases? This means none (zero) of these colluding test-errors are escaping through the test procedures and we are approaching full conformance.

$$P(\text{test} - \text{event} - \text{error}) = 1 - (APE)^N$$

For all test cases, the probability of this happening is:

$$P(\text{test} - \text{event} - \text{error})^M = \left[1 - (APE)^N\right]^M$$

where M is the number of test cases.

It is important to note that the average probability of error (APE) for each implementation is assumed to be the same for each test case. Since this is not always true, it must be approximated. There isn't an easy way to quantify the probability of the error occurring in the specific test so an educated guess must suffice. This unknown value could possibly be computed as:

$$\frac{1}{\text{number of possible code sequences covered by this test case which could be in error}}$$

If the exact probabilities could be computed for each test case, the expression would be:

$$1 - (P_1(\text{test} - \text{error}) \cdot \dots \cdot P_N(\text{test} - \text{error}))$$

Or

$$P(\text{test} - \text{event} - \text{error}) = 1 - \prod_{i=1}^N P_i(\text{test} - \text{error})$$

2.4.1. Example:

N=number of implementations under test = 10, M = number of test cases =20, APE = 0.10 = chances of error in an application on the same test case. We have to guess on this 0.10 because we don't know for sure the exactness of this value.

$$P(\text{conformance}) = (1 - APE^N)^M = \left(1 - \left(\frac{1}{10}\right)^{10}\right)^{20}$$

2.4.2. Finally:

Now this is the chance of any one implementation or the CE being error free with respect to the test criteria. The test criteria are, of course, based on the specification and, since no errors escaped through the test, each of them has passed the **conformance** if the test plan is correct.

2.5. Final Conditions:

- The implementations are developed in a manner that produces random errors – the proof above depends on random errors. Non-random errors invalidate the proof.

The basis of the theorem is that errors happen randomly.

- The appropriate Error Generator application is part of the IOP test. This is required to ensure that the implementation being tested in an interoperability test have the ability to use the same test criteria as those in a **conformance test**.

It is assumed the error generator application implements the same error test as conformance engine.

- The same Test Criteria is used for both.

This should be obviously clear.

- The CE tested against itself.

The CE in the conformance test could be less conformant than the full matrix tested products because how it responds to 'error-conditions' is not tested unless it is tested against itself.

Key Conclusions from this paper:

- One of the most important observations drawn from the probability theorems is the necessity of a well constructed error generator within an interoperability test. The failure to provide one prevents the conformance and interoperability testing from properly establishing the boundaries of the test domain and verifying the products under test truly meet the standards' requirements. A test event which does not use the error generator must be very carefully designed to prevent deployment level interoperability issues from arising.
- The necessity of the error generator points to the fact that testing organizations have to be careful in their communication with organizations developing the products regarding the standard under test. If there is significant dialogue between those developing the implementation and the test organization, they may all make the same error with reference to 'the standard'. If this occurs, the chances are greater that, during the test event, all the implementations may evaluate this 'error' as a 'non-error' and thus evaluating implementations as interoperable and conformant – yet they may not be in reality.
- Assuming the presence of an error generator and well designed testing environment, the probability formula shows that around 10 or more implementations being interoperability or conformance tested gives a significant level of conformance to the implementations from interoperability only test and to the conformance engine in the conformance only test case – for a reasonable APE. However, note that the APE has to be estimated from knowledge of the testing environment and the standard. Significantly less than ten implementations may not “clean” the conformance engine enough in a conformance test. It depends on the value of APE chosen and the number of individual test cases.
- The probability proofs also speak of the value for both conformance testing and interoperability testing for products. While the theorem does not address this directly, one of the assumptions is that the specification will be tested during both types of test. The specification definition is 'loose' in that it could apply to just 'the single standard' or to all 'associated standards on which the main standard is based'. For some test environments there is 'the standard', constructed in a manner that is very independent of other standards – some areas of devices are an example – so test criteria covers only the software or firmware on which the core standard is based. When this

environment is evident, the probability of conformance producing interoperability is much higher than in the environment where 'the standard' is strongly dependent or includes additional 'other software' such as in the case of eBusiness kindred software or firmware. This lower interoperability probability occurs because the test criteria does not cover the 'other software'.

Frequently we do not know if the 'other software' is conformant or interoperable. In this complex environment such as seen in eBusiness software and/or firmware, adding test criteria for this 'other software' would add too much effort to the actual test event and generally it is not done or only done partially. In the end, the conformance engine does not evaluate the 'other software' and does not expose the interoperability problems among the 'other software' area. The converse is true that the probability of interoperability testing achieving a significant level of conformance is less over standard implementations highly dependent on this 'other software'. With this type of standard, use of both conformance testing and interoperability testing is of greater value.

- Must we always test for both conformance and interoperability? The answer is no in some cases. However, this decision is based on efficiency of the test and possible savings for all concerned in the testing effort – implementers and testers alike. It is always best to do both – yet if the error generator is sufficient enough in an interoperability test, it is highly likely one will achieve the same results as doing the additional conformance testing. Also as noted above, conformance testing in some non- eBusiness type software can become close to interoperability or maybe achieve interoperability because there is no 'other software'.

3. GLOSSARY

Definition: COMPOSITE RELATION - $R1 \circ R2$

Let $R1$ be a relation from A to B, $(a, b) \in R1$, and let $R2$ be a relation from B to C, $(b, c) \in R2$, . Then $(a, c) \in R1 \circ R2$, where $a \in A$ and $b \in B$, and $c \in C$.

Definition: CONFORMANCE ENGINE APPLICATION

A conformance engine application is composed of 3 parts:

- a test administrator facility
- a specification mimic that implements at least the parts of the specification to be tested
- an error generator application that produces *error-conditions*, and has the ability to produce non-domain elements to test the *domain* boundaries. These non-domain elements could produce two types of errors: error-conditions or errors the applications do not handle programmatically correctly. We call these later ones *test-errors*.

Definition: CONFORMANCE ENGINE ERROR DEGREE

A conformance engine is said to be error free to degree N on the *specification*, if and only if, it has been tested against N implementations of the *specification* with all implementations producing random errors as they are tested.

Definition: CONFORMANCE TEST

A conformance test of an application shows that the application conforms to the *specification* by interacting with the **conformance engine application**.

Definition: CONFORMANCE TO A SPECIFICATION

An implementation of a *specification* is said to be conformant to the *specification*, IF and only IF, the input *domain*(x, y,..), and the output *range*(a, b,...) of the implementation meet the requirements of the *specification* and the relation, $range = R(domain)$, when R implements the requirements of the *specification*.

This is a normal BLACKBOX arrangement with R as the BLACKBOX, the input being the domain and the output being the range. See Figure 1a.

The dependant and independent variables of the *range* and *domain* may be Boolean, real, integer, documents, sets, etc. --that is, they may be composed of any length bit-stream.

Definition: ERROR-CONDITION

An *error-condition* is a condition in a program where the *domain* of R elicits a known *range*-element of type "error" or warning as the *specification* designates. This is not an error in testing or conformance. This is 'success' because the program is acting as the *specification* designates.

Definition: ERROR GENERATOR APPLICATION

An error generator application produces *error-conditions*, and has the ability to produce non-domain elements to test the *domain* boundaries.

Definition: FULL MATRIX INTEROPERABILITY TEST

A full matrix interoperability test for a set of applications built on a *peer-interoperable specification* shows the applications interact properly -- are peer-interoperable -- each with every other, in a full matrix manner as the *specification* states. Thus it must show that $R2$ initiates and $R1$ responds, $R1(R2(domain))$, and $R1$ initiates and $R2$ responds, $R2(R1(domain))$, properly and both are a subset of *domain* for every pair of applications, whose relations $R1$ through Rn are within the test and an application responds to peer implementation $R(R(domain))$. See Figure 2 and Figure 1b.

Definition: PEER-INTEROPERABLE SPECIFICATION

A *specification* is said to be peer-interoperable, IF and only IF, the input *domain*(x, y, \dots) and the output *range*(a, b, \dots) meet the requirements of the *specification*. It also requires that the relation, R , implements the requirements of the *specification* and *domain* is a superset (\supseteq) or proper superset (\supset) of $R(R(\text{domain}))$.

(The dependant and independent variables of the *range* and *domain* may be Boolean, real, integer, documents, sets, etc. (that is, they may be composed of any length bit-stream.)

Definition: RELATION - R

Suppose R is a relation from A to B . Then R is a set of ordered pairs where each first element comes from A and each second element comes from B . That is, for each pair $a \in A$ and $b \in B$ then $(a, b) \in R$ is read as “ a is R -related to b ”. The domain of a relation R is the set of all first elements of the ordered pairs which belong to R . The range of R is the set of second elements.

Definition: SPECIFICATION

A *specification* is a pre-test agreement among implementers of sufficient detail and exactness which allows evaluation of an implementation as to meeting the *specification*'s conditions.

Note: This definition covers profiling also. It also covers *specifications* that are not standards but are done when two or more companies decide to intercommunicate in a more ad hoc manner. Finally, it may cover all standards such as HTTP, on which, the *specification* is based. This is necessary because we often test a *specification* for conformance or IOP, yet do not test the supporting standards which may not be conformant in the implementations and could potentially cause a problem. For example, if using HTTP, we don't know the HTTP code to make sure that it is conformant to HTTP specifications or has been profiled correctly across all the implementations.

Definition: TEST CRITERIA

The test criteria are the detailed test plan based on the specification under test that is usually composed of many individual test cases.

Definition: TEST-DOMAIN

Test-domain is normally a superset of *Domain* whose purpose is to verify that the relation R is rejecting non-*domain* input elements. (Note: In a well written *specification*, one would expect all *Test-domain* elements minus *domain* elements to produce **error-conditions** and not *test-errors*. However, in distributed applications, *test-errors* or **error-conditions** produced from outside events such as communication errors, communications interruptions do occur.)

Definition: TEST-ERRORS

A **test-error** is when the *domain* of R elicits a condition, for which, there is no element in the *range* because it is undefined in the *specification* or because the application has not been designed properly.

Definition: TESTING-EVENT-ERROR

Type II Errors in which the systems under test:

- Fail to identify a non-conformance error
- Agree that something is not a **test-error** when it actually is
- An error escaping thru the test regime for each test case

[¹] *Schaum's Outlines: Discrete Mathematics*, 2nd ed., Lipschutz, S. and M. Lipson, p. 28, McGraw-Hill, 1997

[²] *Elementary Statistics*, 9th ed. Triola, M., p.381 Pearson Education, 2004



Rik Drummond

CEO and Chief Scientist
[Drummond Group Inc.](#)

As the chief executive officer and chief scientist for Drummond Group Inc. (DGI), Rik Drummond has led the company's technical and research strategies while steering DGI to constant growth and innovation. He is a widely respected authority in the eBusiness industry and has been a driving force in the technical standards bodies and vertical industry groups supporting B2B commerce.

DGI, a global leader in eBusiness and eGov software testing and certification, works with software vendors, industry associations, supply chains and the standards community by conducting interoperability and conformance testing, consulting, publishing related strategic research and developing vertical industry strategies. Founded in 1999, DGI has tested thousands of international software products. DGI's certified products are used in vertical industries such as automotive, consumer product goods, financial services, government, petroleum, pharmaceutical and retail.

Drummond previously served as chair of the Electric Industry's GridWise Architecture Council (www.gridwiseac.org) in the USA, the chairperson for EDIINT Workgroup of IETF (www.ietf.org) which has produced several standards, and was the Workgroup leader for ebXML Messaging v1.0 under OASIS (www.oasis-open.org).