ACTA IMEKO
December 2013, Volume 2, Number 2, 34 – 40
www.imeko.org

GUM conformity of software products - a discussion from a software tester's perspective

Norbert Greif[1], Heike Schrepf[1]

[1] *Physikalisch-Technische Bundesanstalt, Institute Berlin, Abbestraße 2-12, 10587 Berlin, Germany*

ABSTRACT
This paper describes how to assess GUM conformity of software products which claim to offer a GUM-compliant calculation of measurement uncertainties. To bridge the gap between the GUM guideline and the required test specification, an analysis of the GUM from the perspective of software testing is presented. Problems of testability and ambiguity of GUM statements are analysed in detail. The benefit and the limits of the developed validation procedure and test environment are outlined.

Section: RESEARCH PAPER

Keywords: Measurement uncertainty; GUM conformity; measurement software quality; software validation

Citation: Norbert Greif, Heike Schrepf, GUM conformity of software products - a discussion from a software tester's perspective, Acta IMEKO, vol. 2, no. 2, article 7, December 2013, identifier: IMEKO-ACTA-02 (2013)-02-07

Editor: Paolo Carbone, University of Perugia

Received February 12th, 2013; **In final form** November 5th, 2013; **Published** December 2013

Funding: Information not available

Corresponding author: Norbert Greif, e-mail: norbert.greif@ptb.de

1. INTRODUCTION

An increasing number of software products that claim to offer a GUM-compliant calculation of measurement uncertainties are available on the market. In order to ensure that these products perform the calculations in accordance with the GUM [1], a specific validation of the software products with respect to the GUM is necessary.

Additionally, to guarantee comparability of the measurement uncertainties calculated by different software products, a defined comparability of the software products themselves is required. Consequently, a reusable, automated test environment has been developed which supports both a GUM-oriented validation and GUM-related comparisons of different software products by tracing back the product features to the rules and requirements of the GUM (see figure 1). The paper presents the benefit of the test environment, but also the limitations of validation and product comparisons.

To bridge the gap between the GUM guideline and the required test specification, an analysis of the GUM from the perspective of software testing is presented. This detailed analysis of the GUM has uncovered some issues and inconsistencies within the GUM. Included are, for example, non-testable GUM statements, alternative options of implementations of GUM statements, and missing definitions. To ensure unambiguous implementations of the GUM and corresponding explicit test specifications, these ambiguities of the GUM have to be overcome or minimised (see figure 1).

To be sound, first of all, the GUM guideline had to be transformed into a formal specification. For the core clauses of the GUM guideline, such a specification was already presented in [6]. In this paper, the underlying specification is not the focal point. Instead of that, the problems of deriving the specification from the GUM guideline are dealt with.

For example, as outlined in section 3.3, the GUM often allows several computations resulting in differing solutions. The concept of the paper is that each of these possible solutions is considered to be GUM-compliant as long as the solution itself is computed correctly. Thus, several differing nominal results are possible. In the paper, the tester's procedures to deal with different results are described in the clauses called *accepted solutions*.

2. MOTIVATION AND AIM

For several years, the authors have been involved in the evaluation of software products which implement the GUM [3, 4, 5]. Recently, three further software products were comparatively evaluated concerning GUM conformity. During this work, special experience was gathered and an

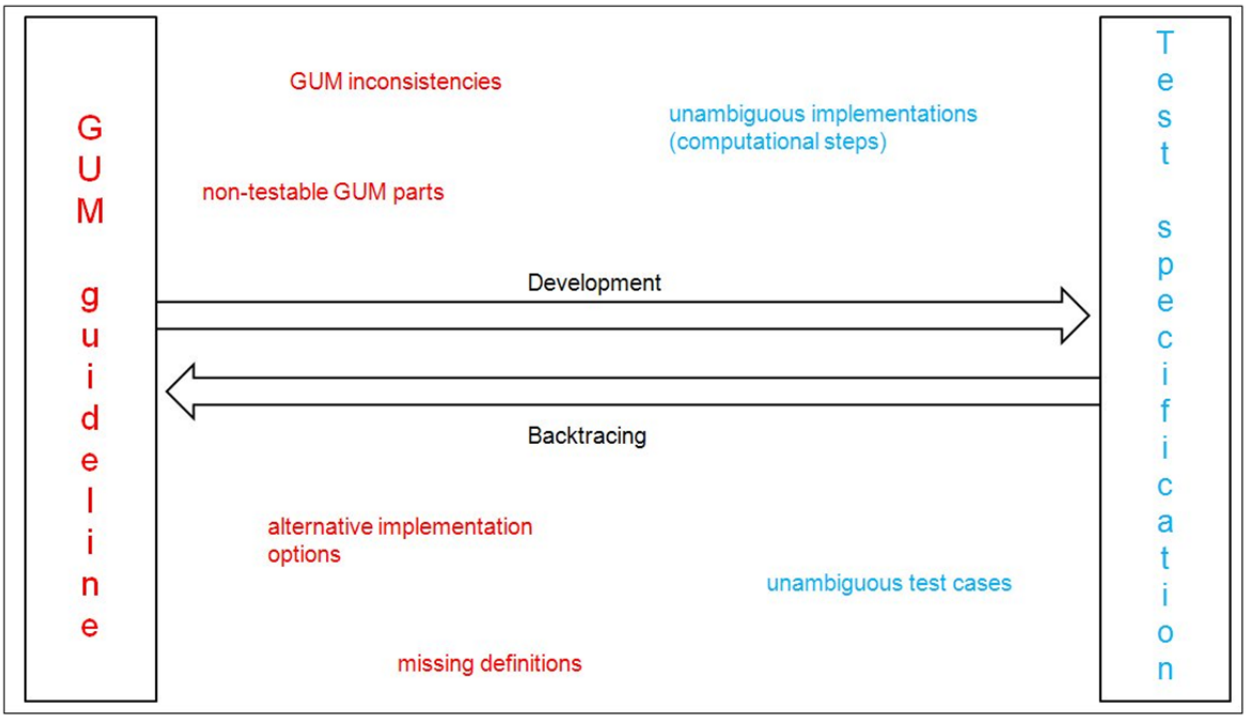


Figure 1. Basic task: Bridging the gap between the ambiguous GUM guideline with inconsistencies or missing definitions, and an explicit test specification.

implementation-oriented view on the GUM has emerged. One aim of the paper is to outline this special experience.

For example, the following fundamental questions have come up regarding an implementation of the GUM:

- **Completeness**
  What does it mean when a software product claims to implement the GUM? Is the GUM completely implementable? Is it possible to reformulate each of the GUM statements so that it can be represented as a computational step?
  Is a pocket calculator already compliant to GUM when it correctly implements only one formula such as the average of repeated observations (as described in GUM 4.2.1)?

- **Correctness**
  What does it mean when a software product claims to be correct?
  Is the product able to calculate the correct values?
  Or is it able to calculate the correct values, round them with a correct rounding procedure, and display them with a correct number of digits?

- **Compliance**
  What does it mean when a software product claims to be compliant (conforming) to the GUM?
  The GUM guideline does not contain a conformity clause. Thus, is it allowed to claim a conformity statement based on completeness and correctness?

Already these few questions lead to one of the core problems of both testing GUM software and comparing GUM test results: The need to trace back each computational step and each test result to a certain well-defined, well-understood and uniformly interpreted GUM statement.

Consequently, traceability should be the precondition for the validation of a specific software product as well as the comparison of different products.

To get repeatable, comparable, and traceable validation results, the questions mentioned above and some further queries have to be answered.

The corresponding answers have an important impact on the set-up of the software test environment. For the software specification step in between, specific introduction and guidance is given in [6].

In summary, the objectives of the work are

- to prove a GUM-compliant calculation of measurement uncertainties (to prove "conformity" of GUM-supporting software products with the GUM guideline);
- to provide comparability of measurement uncertainties calculated using different software products (useful for key comparisons);
- to provide comparability of test results and of the software products themselves.

To achieve these objectives, the main tasks are

- to perform a detailed analysis of the GUM from the software testing and software implementation point of view;
- to trace back each computational step and each test result to a certain well-defined GUM statement;
- to develop a GUM-oriented validation procedure;
- to develop a reusable, automated test environment;
- to support a GUM-related comparison of different software products by tracing back the product features to the requirements of the GUM.

This paper describes the analysis of the GUM (see section 3) and gives a short overview of the validation procedure and the test environment developed (see section 4).

3. ANALYSIS OF THE GUM FROM THE PERSPECTIVE OF SOFTWARE TESTING

In this main section of the paper, some problems of testability and ambiguity of GUM statements are analysed in detail. These issues have a straight influence on the traceability and comparability of validation results belonging to different software products.

In the following, the GUM issues under discussion are classified according to these (non-disjunct) four main categories:

- **Testability of GUM statements**
  Non-testable statements are analysed with respect to additional context information.
- **Strictness of GUM statements**
  Diffuse statements with regard to the possibility to use alternative options are analysed.
- **Ambiguity of GUM statements**
  Ambiguous statements regarding informal wording are discussed.
- **Specific problems**
  Missing GUM statements, GUM inconsistencies and the handling of calculation results not covered by the GUM are analysed.

For each problem, the necessary decisions to guarantee testability, the unambiguous definition of the validation procedure, and the direct consequences for the development of the test environment are derived (cf. the examples with accepted solutions). Some solutions cannot be realised within an automated test environment.

3.1. Testability of GUM statements

The GUM includes a series of statements which are not testable, even in the core clauses 4 through 8. These statements require additional decisions or context information to guarantee the unambiguous definition of the test process. Software packages should be able to ask for the necessary context information. This has not been the case for all software packages validated so far.

Example 1 (GUM 4.2.1, GUM 4.2.3):

The GUM describes the computation of an arithmetic mean for an observation series and allows the use of the computed mean as an estimator for the quantity's value as long as certain preconditions are met. One of these preconditions is the repeatability of observations.

Being allocated a list of observation values, no software package is able to decide whether the required repeatability conditions have been met. If the repeatability condition is to be tested, context information is necessary.

Accepted solution 1: The package asks the user to check the conditions.

Accepted solution 2: The package always assumes certain repeatability conditions. The user manual points out the responsibility of the user.

Example 2 (GUM 3.2.3, GUM 3.2.4, GUM 8.1):

The GUM states that systematic deviations have to be incorporated into the model equation in the form of correction terms.

Being allocated a model equation, no software package is able to decide whether the model equation is complete in this sense.

Accepted solution: The package always assumes completeness of the model equations. The user manual points out the responsibility of the user.

Example 3 (GUM G.2.1):

The GUM states that the output quantity is approximately normally distributed if its variance is "much larger than …".

A software package is not able to compare two values in this informal way.

Accepted solution: The package offers all information necessary to decide on distribution of the output quantity to the user (distributions of all input quantities, their uncertainty contributions, the linearity of the model equation). The user should be able to decide whether the distribution of the output quantity may be understood as normal or is "unknown". In the case "unknown", the package should not calculate and report the expanded uncertainty of that output quantity.

Example 4 (GUM 5.1.2, GUM 5.1.5):

The GUM states that "higher terms (of the Taylor series of the model function) must be negligible".

Should a software package neglect a term when its value is 1/20, 1/100, or 1/1000 of the sum of the low-order terms?

Accepted solution 1: The software package calculates results for both, the standard GUM case (first order of Taylor series), and the sophisticated case (including higher order). If the results for $u_c(y)$ are equal after rounding and shortening, then the higher order terms are negligible.

Accepted solution 2: Alternatively to solution 1, the package can check the linearity of the model equation as long as the results are the same as for solution 1.

Example 5 (GUM F.1.2.1 a) and c)):

The GUM explains that the covariance of two input quantities may be treated as insignificant if certain conditions are met.

The software package is not able to decide whether the required conditions have been met.

Accepted solution 1: The software package calculates both, the result with and without correlation. If the results for $u_c(y)$ are equal after rounding and shortening, then the correlation is negligible.

Accepted solution 2: The software package asks the user to check the correlation values.

3.2. Strictness of GUM statements

The possibility to use alternative options requires additional decisions or assumptions to ensure testability. Such options have to be exercised, for example, in case of the formulation of model equations, or in case of the evaluation of sensitivity coefficients.

Example 6 (GUM 3.1.7, GUM 4.1.1, GUM 4.1.2):

GUM 3.1.7 mentions that the presented concept, although only discussed for scalars, is applicable to vector results, too. However, there is no further treatment.

GUM 4.1.1 presents the model relationship as an equation solved for the scalar output quantity.

GUM 4.1.2 states that the relation between input quantities and output quantities are not necessarily an explicit functional relationship. Instead, an algorithm or a computer program which is able to produce result values y for certain input values x_i may be used.
Accepted solution: The model equation may be represented by an explicit functional relationship or not. It may be formed for scalars or for vector results. Each variant is considered to be compliant.

Example 7 (GUM 5.1.3, GUM 5.1.4):
GUM 5.1.3 describes the sensitivity coefficients as partial derivatives of the output quantity with respect to the input quantities at the point of the estimates of the input quantity values. Note 2 of the same section states that the partial derivatives may be calculated using common numerical methods.
GUM 5.1.4 allows the experimental determination of these sensitivity coefficients.
Accepted solution: Sensitivity coefficients may be determined analytically, numerically, or experimentally. Each variant is considered to be GUM-compliant.

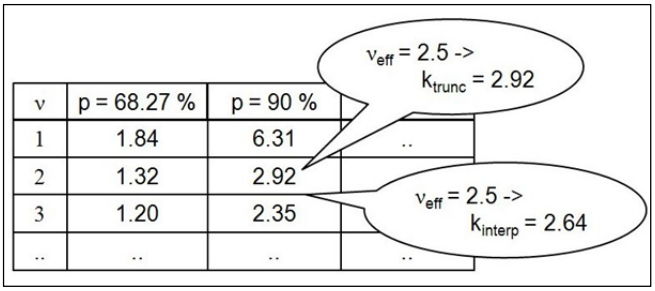
Example 8 (GUM 4.1.4):
GUM 4.1.4 states, that the estimated value of the output quantity is calculated using the estimated values of the input quantities and the model equation.
The following note says that the estimate of the output quantity may also be calculated as the average of several output values, each of them calculated from a set of input values and the model equation.
Accepted solution: The value *y* may be calculated as a function value or as an average of function values.
Each variant is considered to be GUM-compliant. In case of linear models, the results do not differ.

Example 9 (GUM G.4.1, Note 1):
GUM G.4.1 describes the treatment of a degrees-of-freedom value calculated by the Welch-Satterthwaite formula. To derive the coverage factor, two different methods are allowed, interpolation or truncation (cf. figure 2).
Both computations result in significantly differing values.
Accepted solution: Each variant is considered to be GUM-compliant.

3.3. Ambiguity of GUM statements

Ambiguity is caused by informal GUM wording. Usually, the informal wording shall improve readability and comprehensibility of the GUM.

Example 10 (GUM 7.2.6, GUM H):
GUM 7.2.6 explains that the uncertainty should be given with "at most" two significant digits. More digits are allowed to avoid rounding errors in subsequent calculations.
GUM annex H mostly uses two, in some cases only one digit for uncertainty values (H.3, H.5, H.6).
What should the programmer of a GUM package do regarding the question of digits? How should the tester of a GUM package formulate the nominal output for a test case?
Accepted solution: The software package should use two digits by default. It should allow a manual adjustment if necessary.



ν	p = 68.27 %	p = 90 %	
1	1.84	6.31	..
2	1.32	2.92	
3	1.20	2.35	
..	..	..	..

Figure 2. Illustration for example 9 (alternative implementation options).

Example 11 (GUM 7.2.6):
GUM 7.2.6 states that "it may sometimes be appropriate" to round uncertainties up rather than to the nearest digit. Two examples are given: A value like 10.47 should be better rounded up to 11 instead of rounding it to the nearest digit, i.e. 10. In another case, a value like 28.05 should be rounded to the nearest digit, i.e. 28, instead of rounding up to 29.
The GUM obviously uses a rounding principle that is describable as "rounding up or down with a fraction limit somewhere between 0.1 and 0.4, instead of 0.5 as is usual". Since this is not formulated explicitly, each programmer is free to use rounding up or rounding to the nearest digit (and half up).
Accepted solution: Concerning testing, the decision was made to expect rounding to the nearest digit (and half up).

Example 12 (GUM G.6.6):
GUM G.6.6 explains that in certain cases one may use the coverage factor values of 2 (to get a level of confidence of nearly 95%) or 3 (to get nearly 99%). Afterwards, the GUM discusses that in these cases significant over- and underestimations of the confidence interval may occur and that a better estimation may be necessary. The user is recommended to choose a better estimation if the approximation is not sufficient for his purposes.
The question arises whether a GUM package should use the (GUM-compliant) approximation or the (GUM-compliant) better estimation.
Accepted solution(s): The user decides on the kind of distribution of the result quantity using the information delivered by the software package (see Example 3).
If the result quantity may be considered normally distributed:
Accepted solution 1: The user delivers the level of confidence *p* and the package calculates the coverage factor $t(\nu)$ based on a *t*-distribution, or
Accepted solution 2: The user delivers the level of confidence *p* and the package calculates the coverage factor *k* based on a normal distribution, and it delivers the deviation between *k* and $t(\nu)$.
In all other cases:
Accepted solution 1: The distribution is unknown; the package does not calculate the coverage factor.
Accepted solution 2: The distribution is known to the user; the user delivers the coverage factor.

Example 13 (GUM F.2.3.3):
GUM F.2.3.3 discusses the case in which only a minimum and maximum value (and therefore the half width *a*) for an input quantity is available.

The suggestions for the uncertainty of this input quantity vary from $a/\sqrt{3}$ (for a uniform distribution assumption), to $a/\sqrt{6}$ (for a triangular distribution assumption), and to $a/\sqrt{9}$ (for a normal distribution assumption).

GUM S1 6.4.2.1 suggests to assume a uniform distribution, based on the principle of maximum entropy [2].

Accepted solution: The package has to ask the user. He has to decide which assumption holds.

3.4. Specific problems

Finally, some problems regarding missing GUM statements, GUM inconsistencies, and the handling of calculation results not covered by the GUM are considered.

Example 14 (GUM G.4):

The problem of effective degrees of freedom of the output quantity is discussed in relation to the problem of the output quantity's distribution and other aspects (central limit theorem). The given formula (G.2b) and the reference to section GUM 5.1.3 suggest that the formula is valid for uncorrelated input quantities only, but this is not expressed explicitly or discussed in detail.

In particular, there is no explicit prescription not to use formula (G.2b) in case of correlated input quantities.

Accepted solution: The calculation of a degree-of-freedom value for the output quantity in case of correlated input quantities is not considered to be compliant. A value may be given, but its calculation has to be documented, and it has to be marked as outside the GUM scope.

Example 15 (GUM 4.3.8, GUM G, GUM F):

A topic which is discussed very roughly is the usage of input quantities with asymmetric distributions. In this case, GUM statements consist of a single section in the main text (GUM 4.3.8), a short discussion in annex G (GUM G.5.3), and the discussion of a particular case in annex F (GUM F.2.4.4).

The question arises: How should the user deal with asymmetrically distributed input quantities? They cannot be omitted, since GUM does not prohibit their use.

Accepted solution: The distributions of the input quantities do not influence the computation of the value of the output quantity y and the standard measurement uncertainty $u_c(y)$. Displaying y and $u_c(y)$, and omitting $U(y)$ is considered GUM-compliant.

Example 16 (GUM 6, GUM G):

The problem of how to evaluate the expanded uncertainty of an output quantity (which is in practice of greater interest than the standard uncertainty) is only briefly discussed. GUM 6 suggests to use a coverage factor between 2 and 3, and mentions that the selection of a proper value depends on experience or, alternatively, on knowledge about the output quantity's distribution. The details of this discussion take place in annex G.

For testers, the question arises whether a software product is GUM-compliant if it uses an arbitrary coverage factor between 2 and 3 ignoring the statements of annex G.

Accepted solution: The statements of annex G are considered relevant for achieving GUM-compliance.

Example 17 (overall GUM):

It is common sense that a correlation matrix should be checked with respect to its being symmetric and non-negative definite. Most of the GUM packages allow the user to do these checks, but the definiteness is not discussed in the GUM.

Accepted solution 1: The software package checks the non-negative definiteness of the correlation matrix.

Accepted solution 2: The software package does not check the non-negative definiteness of the correlation matrix. Instead of that, before the output of the standard measurement uncertainty $u_c(y)$ of the output quantity, the package checks that the expression for $u_c^2(y)$ is non-negative.

Example 18 (overall GUM):

The experience from the GUM packages that have been validated is that most of these packages compute

- confidence intervals for output quantities with rectangular distribution,
- effective degrees of freedom in case of correlated inputs, and
- confidence intervals for correlated output quantities, irrespective of the fact that the GUM does not prescribe anything in these cases.

Accepted solution: Because these calculation results are not covered by the GUM, they do not belong to a validation of a package with respect to GUM conformity. On the other hand, however, these results are important in practice.

With regard to the test process, testing of these calculations is performed, but the corresponding test cases are marked as "outside GUM conformity testing".

4. OVERVIEW OF THE TEST ENVIRONMENT

In this section of the paper, the test environment as it has been developed for the validation and GUM-related comparison of software products is described very roughly. A schematic overview of the test environment is illustrated in figure 3. A detailed presentation is given in [5].

The following description is restricted to the overall understanding of the test concept and to some aspects which are of importance for the analysis of benefits and the problems mentioned above. Implementation details are omitted.

The objective to validate software products that implement the GUM is best achieved by establishing a well-defined, GUM-oriented test process supported by a reliable technical test environment. The environment itself has to obey certain quality requirements, for example, correctness and completeness. Especially, the test cases must be designed in a way that they generally fit for any GUM-supporting software product under test. Consequently, comparability of certain validation results and after all the comparability of the whole validation process has to be ensured.

To meet these requirements, the test environment consists of the following components:

- **Data model** defining the structure of information necessary for uncertainty calculations and corresponding tests. Main components of the model are the test case identification, the test purpose with classification (cf. figure 3) and GUM reference, the inputs for the software under test, and the nominal outputs which are criteria for the package's results.

- **Set of universal test cases** which do not contain any product-specific or technical information. The test case repository is implemented based on the data model.

Each test case is represented by a separate file with a unique identifier.

- **Test case converter** which translates universal test cases into product-oriented specific ones. The converter needs information about the package to be tested, the underlying operating system, and the test tool which will be used, for example. Depending on the software under test and the validation task, the converter has to filter the test cases.

- **Several sets of product-oriented specific test cases**, each of which belongs to a specific software package to be tested. These test cases contain, for example, package-specific commands, input values, buttons to push and menu items to select.

- **Capture-replay test tools** to operate the test cases and to repeat automatically the overall test process.

The universal and package-specific test cases are arranged concerning a well-defined classification scheme. This classification hierarchy is based on the software quality characteristics defined in the international software standard ISO/IEC 25010 [7].

The respective position of a test case in the hierarchy corresponds to the purpose of the test. In this way, the classification scheme allows a certain control of completeness and traceability of the validation process.

In accordance with figure 4, the main levels of the classification hierarchy are:

- Assignment of the test cases to the set of software quality characteristics according to the software standard ISO/IEC 25010 [7], for example, *functionality, usability*, and *reliability*.
- Subdivision of test cases into positive cases (prove that the GUM is correctly implemented) and negative cases (prove that in case of the non-applicability of the GUM no calculation is carried out).
- Specific subdivisions depending on the value for the first level.

An example for the third classification level is closely connected with the software quality characteristic *functionality* (see figure 4).

In this case, the classification hierarchy represents the detailed calculation steps needed to prove the conformity of the software packages to the core sections and formulas of the GUM. The calculations are split into the following steps (branches of the classification hierarchy, see figure 4):

- Calculations of Type A uncertainties (without correlation of inputs);
- calculations of Type B uncertainties (without correlation of inputs);
- interpretation of model equations and calculation of sensitivity coefficients (SCs in figure 4);
- calculation of values, standard measurement uncertainties, and coverage intervals of output quantities without and with the correlation of input quantities;
- calculation of the correlations between output quantities (vector results);
- calculation of the examples from GUM Annex H.

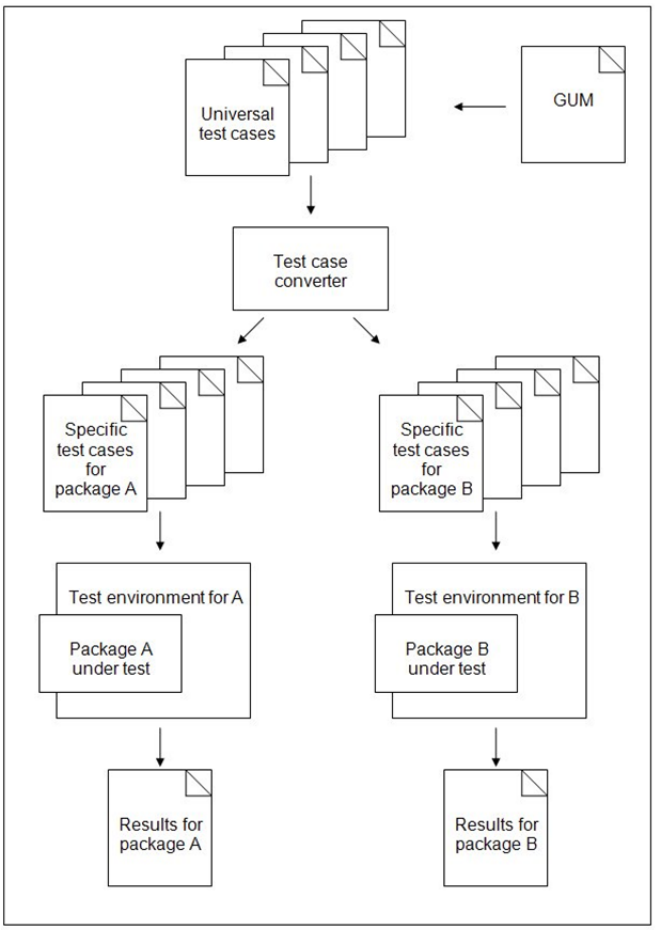


Figure 3. Schematic overview of the test environment.

For each of these calculation steps, further classification levels depending on the degree of complexity of the test cases can be defined. Normally, we use between five and nine classification levels.

In addition to the quality characteristic *functionality*, the characteristics *usability* and *reliability* were used to design and implement test cases. In future, the characteristic *efficiency* might become relevant to include response time evaluations of Monte Carlo simulation engines.

5. CONCLUSIONS

A number of software packages which claim to implement the GUM are on the market. However, they differ in functionality and the have deficiencies which are not obvious. Thus, a validation of these packages with respect to the GUM is necessary.

The PTB test environment has been used successfully to validate and compare three different GUM-supporting software packages.

To bridge the gap between the GUM guideline and the explicit test specification, a detailed analysis of the GUM from a tester's perspective and certain decisions regarding the test process (cf. the accepted solutions of the examples in section 3)

were necessary. Based on the results of this analysis, unambiguous and detailed test cases could be developed. The benefits of the test environment and the validation procedure are:

- General procedure usable for any GUM-supporting software product;
- automated and reusable process;
- comparability of the validation procedure and, especially, of the validation results;
- automated documentation process.

However, there are also limitations in the validation procedure, and in the process of product comparison.

The current procedure includes sections 5.5 to 5.8, and 6 of [2], but does not consider sections 5.9, 5.10, and 7 (Monte Carlo simulations), and does not regard the handling of complex numbers. The general limitation is, that several obstructive characteristics of GUM statements (with regard to software testing), such as ambiguities, missing or inexact specifications/definitions, do restrict the applicability and the objectiveness of the test environment.

Thus, some of the accepted solutions cannot be realised within an automated test environment.

Concerning the software quality characteristics, up to now, the validation procedure does not include *efficiency* testing (e.g. duration of Monte Carlo simulations).

In principle, the test environment is prepared to realise the extensions mentioned above. Some extensions concerning Monte Carlo simulations and vector results are already under construction.

The work reported reveal some problems regarding the objectives of testing GUM-supporting software products and the corresponding GUM statements. These problems, for example, GUM inconsistencies or ambiguities, have to be minimised. Directly, they concern the implementation of GUM-supporting software products and the corresponding product validations. The further discussion of these problems would enhance the traceability of implementation and validation results to the GUM and the comparability of uncertainty calculations performed by different software products.

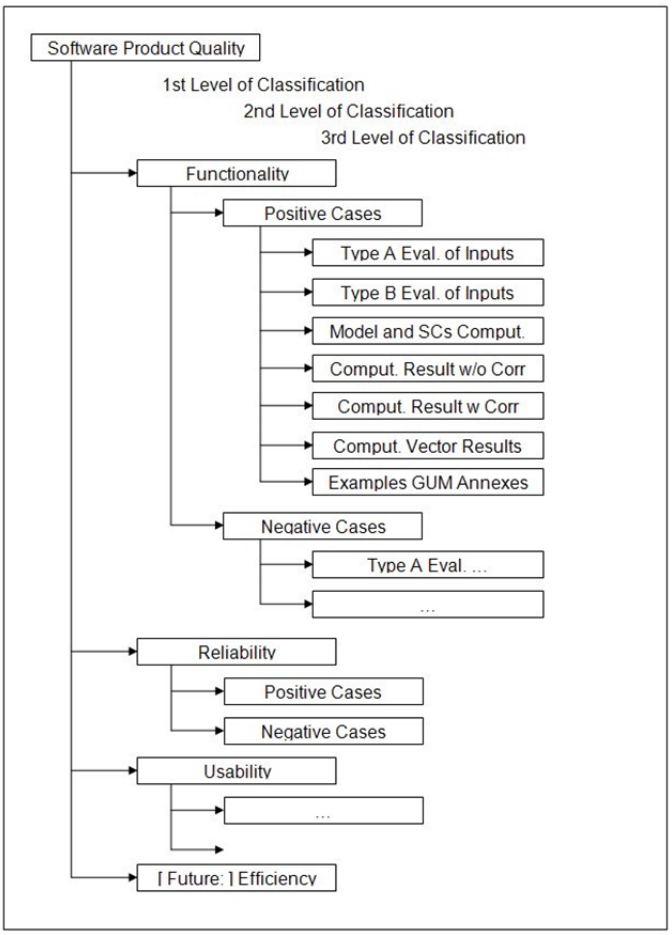


Figure 4. Classification hierarchy of test cases (extract).

REFERENCES

[1] ISO/IEC Guide 98-3:2008, Uncertainty of measurement - Part 3: Guide to the expression of uncertainty in measurement, 2008.

[2] ISO/IEC Guide 98-3:2008/Suppl 1:2008, Propagation of distributions using a Monte Carlo method, 2008.

[3] N. Greif, H. Schrepf, D. Richter, Software validation in metrology: A case study for a GUM-supporting software, Measurement, Volume 39, 2006, pp. 849-855.

[4] N. Greif, H. Schrepf, Validierung von Software zur Bestimmung von Messunsicherheiten, VDI-Berichte 1947, Messunsicherheit praxisgerecht bestimmen, VDI, 2006, pp. 409-418.

[5] N. Greif, H. Schrepf, V. Hartmann, G. Kilz, A test environment for GUM conformity tests, Physikalisch-Technische Bundesanstalt (PTB), Braunschweig und Berlin, PTB Report, to appear, 2013.

[6] M. G. Cox, P. M. Harris, I. M. Smith, Software specification for uncertainty evaluation, NPL Report MS 7, March, 2010.

[7] ISO/IEC 25010:2011, Systems and software engineering - System and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, 2011.