Software Support for Metrology
Best Practice Guide No. 1

Measurement System Validation: Validation of Measurement Software

Brian Wichmann

April 2000

Software Support for Metrology
Best Practice Guide No. 1

Measurement System Validation:
Validation of Measurement Software

Brian Wichmann
with Robin Barker, Maurice Cox, Peter Harris
Centre for Mathematics and Scientific Computing

April 2000

ABSTRACT

The increasing use of software within measurement systems implies that the validation of such software must be considered. This Guide addresses the validation both from the perspective of the user and the supplier. The complete Guide consists of a Management Overview and a Technical Application.

ISSN 1471–4124

Authorised by Dr Dave Rayner,
Head of the Centre for Mathematics and Scientific Computing

National Physical Laboratory,
Queens Road, Teddington, Middlesex, United Kingdom TW11 0LW

Preface

The use of software in measurement systems has dramatically increased in the last few years, making many devices easier to use, more reliable *and* more accurate. However, the hidden complexity within the software is a potential source of undetected errors. Both users and developments of such systems must be aware of the risks involved and take appropriate precautions.

In this Guide, we consider the implications of the use of software in measurement systems. Such software could be embedded within the measurement system, be provided by the system supplier but be separate from the measurement system, or could be developed separately to work with one or many types of measurement system. The key issue is to preserve the integrity of the measurement process.

Although users of measurement systems have a different perspective than the developers of measurement systems, just one Guide is provided. The reason for this is that the user needs to understand what the developer can reasonably provide to demonstrate the integrity of the measurements. Equally, the developer needs to be aware of the legitimate concerns of the users in order to provide assurance in a manner that can be accepted by the users. In a competitive market, a consistent approach to quality assurance for measurement software is required.

This Guide is provided in two parts. The first part is a Management Overview of the area which defines a framework for an approach to quality assurance. This framework is based upon a risk assessment which is then used to determine the most appropriate validation techniques to be applied.

The second part of the Guide provides the technical details of the validation techniques recommended. This material is supported by many references, research papers and related documents which can assist in the application of the techniques.

For convenience, the complete material of the Guide is only available in electronic format. Users can decide what parts, if any, can be printed. Alternatively, the material can be inspected by the use of an Internet Browser or Adobe's Acrobat reader. The Management Overview is available in printed form, without *the technical part* or *the appendices*.

This Guide is based upon a previous guide that was published under the Competing Precisely programme. However, this new Guide has a wider scope and takes into account all the comments made on the previous guide.

Acknowledgements

Thanks to Bernard Chorley and Graeme Parkin for support and help with this project. Brian Bibb (Elektra Oncology Systems Ltd) provided some useful references which greatly assisted this work.

Comments are acknowledged from: Dr Peter Vaswani (UKAS), Dr John Wilson (BMTA), Brian Bibb, John Stockton, Bill Lyons (Claude Lyons Ltd), Dr Richard Mellish (Medical Devices Agency).

Examples have been provided by Ben Hughes (NPL, CLM), Nick Fletcher (NPL, CEM) and Phil Cosgriff (Pilgrim Health NHS Trust).

This guide was produced under the *Software Support for Metrology* programme, which is managed on behalf of the DTI by the National Physical Laboratory. NPL is the UK's centre for measurement standards, science and technology.

For more information on the *Software Support for Metrology* programme, contact Dave Rayner on 020 8943 7040 (e-mail: dave.rayner@npl.co.uk) or by post to National Physical Laboratory, Queens Road, Teddington, Middlesex, United Kingdom TW11 0LW.

Contents

Part I

Management Overview

Chapter 1

Introduction

Almost all of the current generation of measurement systems contain a significant amount of software. Since it is hard to quantify the reliability or quality of such software, two questions immediately arise:

- As a *user* of such a system, how can I be assured that the software is of a sufficient standard to justify its use?
- As a *supplier* of such software, what development techniques should I use, and how can I assure my customers of the quality of the resulting software?

This Guide addresses these two questions. The intended readership are those responsible for software in measurement systems and those using such software. Since progress depends on both the user and supplier, this Guide merges both views, but distinguishes them by reference to *user* and *supplier* when necessary.

Software written as a research project or to demonstrate the feasibility of a new form of measurement is excluded from the scope of this Guide.

The Guide surveys current good practice in software engineering and relates this practice to applications involving measurement systems. Known pitfalls are illustrated with suggested means of avoiding them.

The general approach is a four stage process as follows:

1. An analysis of the physical process upon which the measurement system is based.
2. A risk assessment based upon a model of a measurement system with its software.
3. An assessment of integrity required on the software, based upon the risk assessment (called the **Software Integrity Level**).
4. Guidance on the software engineering methods to be employed determined by the **Software Integrity Level**.

It must be emphasised that there is no simple universal method (silver bullet) for producing correct software and therefore skill, sound technical judgement and care are required. Moreover, if it is essential for the quality of the software to be demonstrated to a third party, then convincing evidence is needed which

should be planned by the *supplier* as an integral part of the software development process.

To aid in the application of this Guide, detailed technical material and check lists are provided in part II.

To avoid complexity in the wording of this Guide, it is assumed that the software is already in existence. It is clear that use could be made of the Guide during the development, but it is left to the reader to formulate its use in that context.

No consideration has been given here of the possibility of standardising this material, or obtaining some formal status for it.

The use of software either within or in conjunction with a measurement system can provide additional functions in a very cost-effective manner. Moreover, some measurement systems cannot function without software. Hence, it is not surprising that there is an exponential growth of software in this area. Unfortunately these changes can give rise to problems in ensuring that the software is of an appropriate quality.

The problem with software is largely one of unexpected complexity. Software embedded with a measurement system could be inside just one ROM chip and yet consist of 1Mbyte of software. Software of such a size is well beyond that for which one can attain virtually 100% confidence. This implies that one has to accept that there is a possibility of errors occurring in such software.

An area in which there has been a substantial effort to remove software errors is in safety applications, and hence the methods used and specified in safety-critical systems are used here as an indication of what might be achievable, typically at a significant cost. For general advice in this area, see [HF-GUIDE].

An example area in which very high standards are required in software production is that for airborne flight-critical software. The costs for producing such software can easily be one man-day per machine instruction — obviously too demanding for almost all software within measurement systems. Hence the main objective behind this Guide is to strike a balance between development cost and the proven quality of the software.

This Guide does *not* cover the application of measurement systems to safety applications. It is hoped that this issue might be handled in a revised version of this Guide produced under a subsequent SS*f*M programme.

The main approach taken here is one of *risk assessment* as a means of determining the most appropriate level of rigour (and cost) that should be applied in a specific context.

A major problem to be faced with software is that the failure modes are quite different than with a simple instrument without software. An example of this is that of the non-linear behaviour of software in contrast to simple measuring devices — see **Software is non-linear** in section A.1.

Chapter 2

Standards and Legal Requirements

There are a number of standards which specify requirements for software in measurement systems which are collected together here with an indication of the issues to be covered by this Guide. The more relevant standards appear first. Standards are important since they are an agreement between the *user* and *supplier* communities.

ISO/IEC 17025. In due course, this new standard [ISO 17025] will replace both ISO Guide 25 and EN 45001. The vital paragraph on computers is as follows:

> *5.4.7.2 When computers or automated equipment are used for the acquisition, processing, recording, reporting, storage or retrieval of test or calibration data, the laboratory shall ensure that:*
>
> *a) computer software developed by the user is documented in sufficient detail and suitably validated as being adequate for use;*
>
> *b) procedures are established and implemented for protecting the data; such procedures shall include, but not be limited to, integrity and confidentiality of data entry or collection, data storage, data transmission and data processing;*
>
> *c) computers and automated equipment are maintained to ensure proper functioning and are provided with the environmental and operating conditions necessary to maintain the integrity of test and calibration data.*
>
> *NOTE Commercial off the shelf software, (e.g., word processing, database and statistical programmes) in general use within its designed application range may be considered sufficiently validated. However, laboratory software configuration/modifications should be validated as in 5.4.7.2a.*

One can see that the main emphasis is upon software developed by the user rather that the software embedded within an instrument or that within standard packages. Although the major risk may well be with

user-developed software, it should not be assumed that other software is without any risk. It would be reasonable to make no checks on word processing software, but just inspect the resulting documents. The same is not true of Spreadsheets where unstable (or less than ideal) algorithms have been used which cannot be easily so easily detected[R-25]. Similarly, simple embedded software should not be a serious risk[SMART]. However, complex software is bound to be a risk since one cannot expect all possible uses to have been checked by the supplier.

ISO/IEC Guide 25. This is the ISO equivalent [ISO 25] of the M10 (see below). Paragraph 10.6 states:

Calculations and data transfers shall be subject to appropriate checks.

Paragraph 10.7 states:

Where computers or automated equipment are used for the capture, processing, manipulation, recording, reporting, storage or retrieval of calibration or test data, the laboratory shall ensure that:

1. *the requirements of this Guide are complied with;*
2. *computer software is documented and adequate for use;*
3. *procedures are established and implemented for protecting the integrity of data; such procedures shall include, but not be limited to, integrity of data entry or capture, data storage, data transmission and data processing;*
4. *computer and automated equipment is maintained to ensure proper functioning and provided with the environmental and operating conditions necessary to maintain the integrity of calibration and test data;*
5. *it[the laboratory] establishes and implements appropriate procedures for the maintenance of security of data including the prevention of unauthorized access to, and the unauthorized amendment of, computer records.*

This Guide gives the most detailed indication of the requirements, and hence is the most useful for both development and assessment.

An IT interpretation of Guide 25 is provided by ISO/IEC TR 13233 [ISO 13233]; however this interpretation does not specifically deal with software in measurement systems, although it could be extended to do so in the future.

EN45001. This standard [EN45001] is the European equivalent to the previous standard. Section 5.4.1 states that: *All calculation and data transfers shall be subject to appropriate checks. Where results are derived by electronic data processing techniques, the reliability and stability of the system shall be such that the accuracy of the results is not affected. The system shall be able to detect malfunctions during programme execution and take appropriate action.*

The standard gives a different gloss on the same area. Here, (numerical) stability and reliability are mentioned, but security and integrity are not. Again, following this Guide should ensure compliance with this standard.

M10. This document is the UKAS laboratory accreditation standard [M10]. Section 8.6 states: *The Laboratory shall establish procedures when using computer data processing to ensure that the collection, entry, processing, storage, or transmission of calibration or test data is in accordance with the requirements of this Standard.* Section 8.7 states: *Calculations and data transfers shall be subject to appropriate checks.*

In practice, these requirements are interpreted by the UKAS Assessor. It is hoped that this Guide will aid this interpretation and reduce the risk of the Assessor taking a different view from the Laboratory.

Legal Metrology. The WELMEC documents summarise the position for such applications [WELMEC-2.3, WELMEC-7.1]. The measuring instrument in this class includes weighing machines, gas/water/electricity meters, breath analysers and exhaust gas analysers. The requirements here derive from the EU Directive 90/384/EEC which has three relevant parts as follows:

1. Annex I, No 8.1: *Design and construction of the instruments shall be such that the instruments will preserve their metrological qualities when properly used and installed, and when used in an environment for which they are intended....*
2. Annex I, No 8.5: *The instruments shall have no characteristics likely to facilitate fraudulent use, whereas possibilities for unintentional misuse shall be minimal. Components that may not be dismantled or adjusted by the user shall be secured against such actions.*
3. Annex II, No 1.7: *The applicant shall keep the notified body that has issued the EC type-approval certificate informed of any modification to the approved type....*

Clearly, only part of these requirements is relevant to the software of measurement systems in general. The WELMEC Guides derives three specific requirements for software from the above Directives as follows:

1. Section 3.1: *The legally relevant software shall be protected against intentional changes with common software tools.*
2. Section 3.2: *Interfaces between the legally relevant software and the software parts not subject to legal control shall be protective.*
3. Section 3.3: *There must be a software identification, comprising the legally relevant program parts and parameters, which is capable of being confirmed at verification.*

In the context of measurement systems not within the ambit of legal requirements, there are two important principles to be noted from the above:

- The handling of the basic measurement data should be of demonstrably high integrity.
- The software should be properly identified (this arises from configuration control with ISO 9001 [ISO 9001, NIS37], in any case, but there is no requirement that ISO 9001 is applied to such machines).

IEC 601-1-4. The standard covers the software in medical devices [IEC 601] and is used both in Europe to support a Directive and by the FDA in the USA [FDA]. The standard is based upon risk assessment with the software engineering based upon ISO 9000-3 [ISO 9000-3]. The flavour of the standard can be judged from a few key extracts below, with those parts relevant to this Guide being:

1. Section 52.204.3.1.2: *Hazards shall be identified for all reasonably foreseeable circumstances including: normal use; incorrect use.*
2. Section 52.204.3.1.6: *Matters considered shall include, as appropriate: compatibility of system components, including hardware and software; user interface, including command language, warning and error messages; accuracy of translation of text used in the user interface and 'instructions for use'; data protection from human intentional or unintentional causes; risk/benefit criteria; third party software.*
3. Section 52.204.4.4: *Risk control methods shall be directed at the cause of the hazard (e.g. by reducing its likelihood) or by introducing protective measures which operate when the cause of the hazard is present, or both, using the following priority: inherent safe design; protective measures including alarms; adequate user information on the residual risk.*
4. Section 52.207.3: *Where appropriate the specification shall include requirements for: allocation of risk control measures to subsystems and components of the system; redundancy; diversity; failure rates and modes of components; diagnostic coverage; common cause failures; systematic failures; test interval and duration; maintainability; protection from human intentional or unintentional causes.*
5. Section 52.208.2: *Where appropriate, requirements shall be specified for: software development methods; electronic hardware; computer aided software engineering (CASE) tools; sensors; human-system interface; energy sources; environmental conditions; programming language; third party software.*

It can be seen that this standard is mainly system-oriented and does not have very much to state about the software issues. However, the key message is that the level of criticality of the software must be assessed, and the best engineering solution may well be to ensure the software is not very critical. This standard covers only instruments which are on-line to the patient as opposed used to analyse specimens from a patient (say). Not all medical applications could be regarded as "measurement systems", and therefore the relevance of this guide needs to be considered.

DO-178B. This is the standard [DO-178B] for civil avionics safety-critical software. It is not directly relevant. However, if an instrument were flight-critical, then any software contained within it would need to comply with this standard. In practice, instruments are replicated using diverse technologies and hence are not often flight-critical. This standard is very comprehensive, and specifies an entire software development process, including details on the exact amount of testing to be applied.

The conclusion for this Guide is that this standard is only relevant for very high-risk contexts in which it is thought appropriate to apply the most demanding software engineering techniques. This standard can be taken as an ideal goal, not achievable in practice, due to resource constraints.

IEC 61508. This standard is a generic standard for all safety-critical applications [IEC 61508]. In contrast to the previous standard, this one allows for many methods of compliance. A catalogue is provided in Part 3 of the standard which handles the software issues. This catalogue is used here as a means of selecting specific methods that may be appropriate in some contexts. This generic standard is less demanding than the previous one, and hence could be applied without the same demands on resources. Guidelines for use within the motor industry have been developed from this standard [MISRA].

The conclusion for this Guide is that this standard is not directly relevant (unless the measurement system is used in safety applications), but could be applied in specific contexts. The catalogue of techniques provides a reference point to a wide variety of software engineering methods. For an analysis of this standard for accreditation and certification, see [UKAS-61508].

A very informative report [NORDTEST], that has many parallels to this one, undertakes an assessment of devices which could be a measurement system based upon an early draft of IEC 1508 (now 61508).

For a detailed research study of assessing instruments for safety application by means of a worked example, see the SMART Reliability study [SMART]. This study was based upon (an earlier edition of) this Guide, but enhanced to reflect the safety requirements. The issue of the qualification of SMART instruments in safety applications is noted as a research topic in a Health and Safety Commission report [HSC].

The conclusion from this survey of the standards is that most of their requirements on software are broadly similar and that aiming to meet all the requirements is a reasonable way of proceeding. One exception to this is that the very demanding requirements in DO-178B cannot be realistically merged with the others. Hence, if a measurement system is required to meet the most demanding levels of DO-178B, then that cannot be expected of an instrument designed to satisfy the other standards mentioned here. Thus this Guide aims to provide advice on producing software which will satisfy any of these standards, with the exception of DO-178B (levels A and B) and IEC 61508.

Chapter 3

From Physics to Measurement

A measuring device is based, of course, on physical properties. However, the underlying physics can be very simple and well-established, or complex, or not well-established. The software may need model a complex physical process and therefore require a demanding validation process.

3.1 Specification of the physical processes

The issues which arise from the modelling of physical processes are considered in the Modelling Theme of the SS*f*M programme [SSfM-Model] but are summarised here.

In a perfect world, building a mathematical model and solving it would be a well-defined process. It is the fact that the model is an approximation of the real world that causes difficulties.

Two questions have to be addressed (essentially by the *supplier*, but the *user* needs to consider this also to appreciate the risk involved):

- Does the software specification provide a faithful implementation of the physics underlying the operation of the measurement system?
- Does the software implement the specification correctly?

In particular, the first question can be quite difficult to answer and requires input from an expert physicist (rather than mathematician or software engineer).

3.2 Producing the model

Some of the issues to be considered:

1. The use of a mathematical model to provide the conceptual interface between the metrology and the software. The model must includes descriptions of

- the measured quantities,
- the errors in the measured quantities,
- the measurement result(s),
- the relationship between measured quantities and measurement result(s).

2. There is a wide range of complexities of mathematical model. However, the model is generally required to be sufficiently complex to reflect accurately the physical situation but also sufficiently simple for the software computations based on the model to be feasible.

3. An estimate of the measurement result is obtained by solving the model in some sense taking into account the defined error structure. Again, the complexity of this process can vary widely.

4. Approximations and assumptions are often made when writing down the mathematical model. Models are generally approximate (certainly for empirical models but also for physical models), and model validation is a means to understand and quantify any modelling error. Assumptions are also made about the errors in the measured quantities (particularly with regard to the underlying probability distributions) which then affects the solution process and the accuracy of the measurement result. Metrics such as *bias* and *efficiency*[1] are often used to measure and compare the performance of different solution algorithms that reflect different sets of assumptions.

5. The procedure for establishing the internal data, such as calibration constants, might be considerably more complicated that the procedure implemented by the measurement system's software.

6. The need to consider the affect of the control software. For example, the control software might make the model more complicated by introducing additional measurement error.

3.3 Validating the implementation of the model

Having established a software specification, the second question is generally more clear-cut. It is the subject of the SS*f*M numerical software testing project, although there are perhaps some particular issues that arise for software that is embedded within a measurement system:

- The requirements on the software, for example, numerical accuracy requirements, are typically dictated by the physical application.
- The software can be tested in isolation from control software and basic instrument.

[1]These two measures relate to the skewness and spread (respectively) of the resulting measurement caused by the error in the raw measurement data, for which an uncertainty estimate is produced; see forthcoming report on Uncertainty.

- In common with other SS*f*M projects, when model validation is required, we advocate the use of Monte-Carlo simulation to validate the mathematical model.

We have several categories of measurement system technology in this area:

Validated physics: This means that the physical model is widely understood and used. Any discrepancy in practice between the model and the observations are within the tolerance of the accuracy specified for the measurement system. Obviously, it should be possible to quote a reference to the physical model in the reviewed literature.

Physics can be modelled: The physics can be modelled, but there is no widely agreed literature supporting such a model as the basis of a measurement system. Hence, for high assurance measurement, it is necessary to validate the model being used. See the relevant part of the SS*f*M programme for research and information in this area [SSfM-Model].

Underlying model uncertain (pragmatic approach): In some situations, there is no precise physical model which can be expressed mathematically. A pragmatic approach may be needed, such as calibration at a number of points together with interpolation when in use. The implications of such a method needs to be understood. For instance, it is necessary to ensure that any non-linear effects are properly taken into account. (See 3.7 for a problem in this area.)

Physics being researched: In this situation, even the model is still be be determined. Measurement systems in this category are not included within the scope of the Guide.

The *supplier* should be able to inform the *user* which of the above categories applies to a specific measurement system.

Given a measurement system as a black-box, then the physical process being used might not be apparent. Obviously, in that case, the measurement system can only be tested as a whole and therefore the question of the quality of the software does not arise directly. This implies that any errors that may be presence in the software could only be observed by appropriate use of the measurement system.

3.4 A measurement system model

In order to provide a framework for the discussion of the software for a measurement system, we present here a simple model. The components of the model in Figure 3.1 are as follows:

Basic instrument. The basic instrument contains no software. It performs functions according to the control logic and provides output. This instrument itself is outside the scope of this Guide, but it is essential that the properties of the instrument are understood in order to undertake an effective appraisal of the software. The basic instrument contains sensors and appropriate analogue/digital converters.

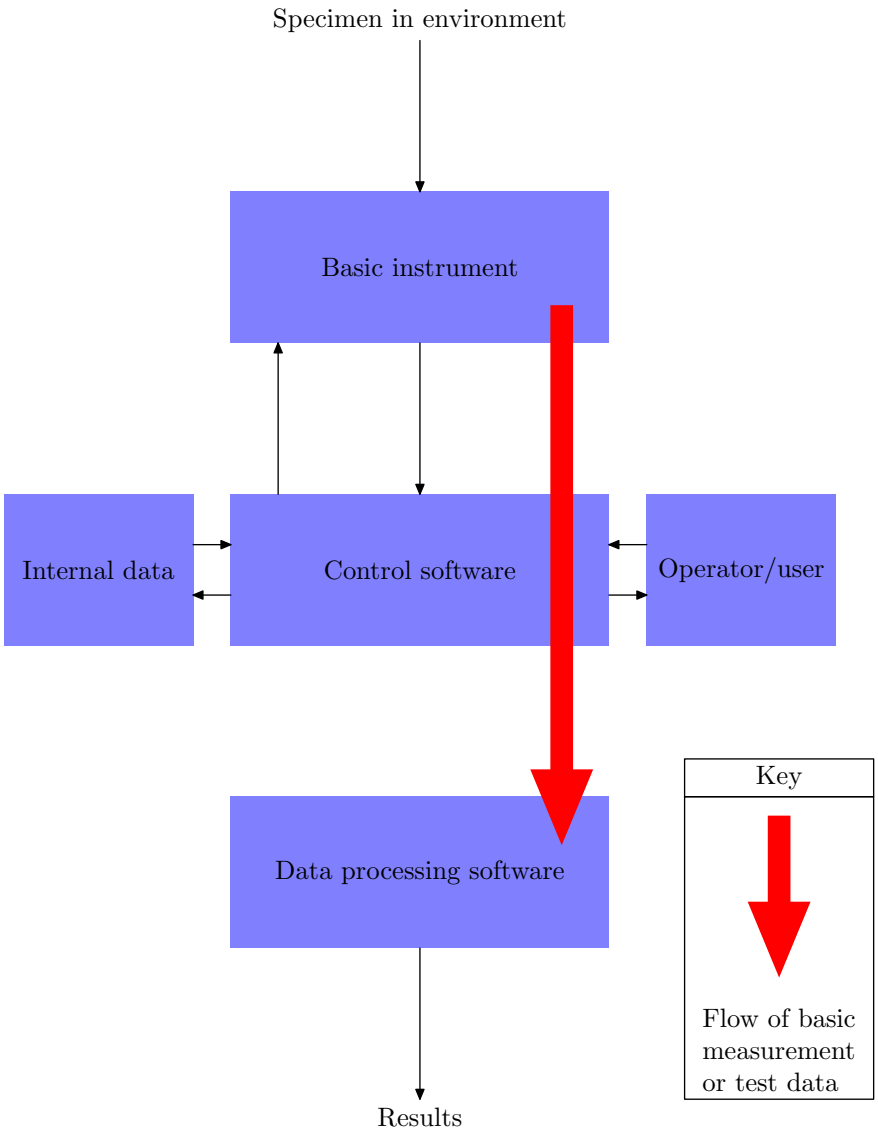

Figure 3.1: A model of a measurement system

Control software. This software processes output from the basic instrument for the purpose of undertaking control actions.

Data processing software. This software performs a series of calculations on data from the basic instrument, perhaps in conjunction with the control software, to produce the main output from the measurement system. (In the case of complex measurement systems, like Coordinate Measuring Machines, the data processing software provided can include a programming language to facilitate complex and automatic control.)

Internal data. This data is held internally to the measurement system. A typical example would be calibration data. Another example might be a clock which could be used to 'time-out' a calibration.

Operator/user. In some cases, there is an extended dialogue with the user which implies that the control function can be quite complex. This dialogue could be automated via some additional software (which therefore could be included or excluded from this model).

In this model, we are not concerned about the location of the software. For instance, the control software could be embedded within the measurement system, but the data processing software could be in a PC or workstation. It is even possible for the subsequent data processing to involve several computers via a Laboratory Information Management System (LIMS). In applying this Guide, you may have a choice in deciding where to draw the line at the bottom of this diagram. For instance, one could decide to include or exclude a LIMS. If the LIMS is considered, then reference [LIMS] on a medical application provides some useful insights. The integrity of the production of the test/measurement certification should not be forgotten [NIS41].

The basic measurement/test data is a key element in this structure. The major requirement is to show that the processing and entire flow of this data has suitable integrity. Note that in the area of legal metrology, the basic measurement data is converted into money (say, in a petrol pump) and this therefore has the same status as the basic measurement data.

It is important to decide the scope of the measurement system. For instance, in DNA analysis, samples are analysed for the Base-Pairs in the DNA. Subsequently, the Base-Pairs found are compared with other sample or population statistics to prepare a statement for the Courts. In this context, it seems that only the determination of the Base-Pairs could be regarded as measurement system software.

3.5 Abstraction at several levels

A measurement system model can exist at several levels and the appropriate level of abstraction must be chosen to fit the objectives of validation. The different levels are illustrated with reference to a Coordinate Measuring Machine (CMM), see figure 3.2.

The basic machine just produces (x, y, z) coordinates from some artifact. Hence the processing software within the basic instrument is probably quite simple — just performing a small calibration correction to the raw data. On

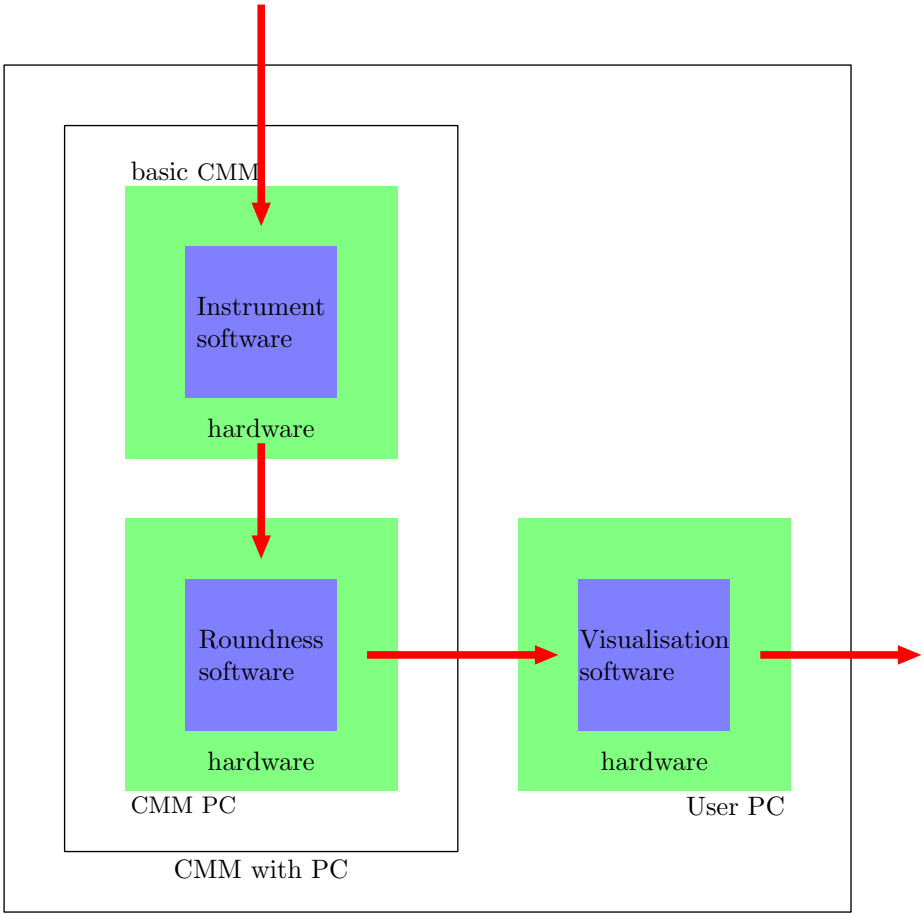

Figure 3.2: Levels of measurement system abstraction

the other hand, the control software could be quite complex if the measurement probe is to be moved round a complex object. One would like to be assured that this complex control software cannot result in false measurements.

Now consider another measurement system which measures artifacts to determine their sphericity (departure from a sphere). This measurement system is actually a CMM with some additional software running on a PC. We now have a measurement system with complex data processing software which needs to be carefully checked if the measurement of the departure from a sphere is to be authenticated.

We can envisage a further level where the sphericity data is processed by visualisation software on the user's PC. The complete system from CMM probe to images on the user's PC is a measurement system and it may be appropriate to validate the system including the visualisation software.

All three examples satisfy the model of a measurement system used in this Guide. However, it is clear that the implications are very different. Hence it is important that the appropriate level of abstraction is chosen for the application in mind.

3.6 Example 1: Druck's pressure transducer

As part of the research programme funded by the Nuclear Industry, NPL undertook a study on the reliability of SMART instruments [SMART]. This study was specifically to investigate the software aspects of reliability which required that the software design and implementation was analysed. To this end, Druck Ltd assisted NPL in providing full details of the software. NPL was effectively taking the role of the *user* (on behalf of the nuclear industry with a safety application), while Druck was the *supplier*.

The results of this study can be summarised as follows:

1. The physical model used was simple.
2. Modelling the physics within the instrument was numerically sound.
3. The software, written in C, passed all the tests run by NPL.
4. There was no reason why such an instrument, using software, should not be used in a safety application.

NPL had complete visibility of the software, and the results satisfied the specification of the instrument.

3.7 Example 2: A vacuum pressure measurement device

NPL has had experience in the past with a pressure measurement device in the vacuum range which works by observing the slowing down of a rotating sphere. For low pressures, the rate of deceleration of the sphere is proportional to the pressure. However, at higher pressures, which is within the range of the device, the linear relationship breaks down. Software within the device compensates for this. However, NPL observed that this compensation was not correct in parts of

the range, thus giving rise to a 'kink' in the graph of the actual pressure against the observed pressure.

Several lessons can be learned from this example:

1. Compensation for a non-linear response should be specified by the measurement system supplier in case the user wishes to take note of any potential risk that this might entail.
2. The dangers are clearly greater if high accuracy is required.
3. If the 'raw' data from the measurement system is available (in this case, the observed deceleration), then the compensation can be done externally or checked separately.
4. Even if a spot check is made for a single high value, the compensation error might not be detected.

Chapter 4

Assessing the Risks

In this section, we undertake an analysis of a measurement system and its related software, the purpose of which is to make an objective assessment of the likely risks associated with a software error. For a general discussion on risk, see [RISKS]. The primary responsibility for the risk assessment must rest with the *user*, but for the user to undertake this in a meaningful manner, information from the *supplier* is often vital.

4.1 Risk factors

The first step in undertaking the risk assessment is to characterise the measurement system according to aspects which influence the risk. Hence we list those aspects below.

Criticality of Usage. It is clear that the usage of some measurement systems is more critical than others. For instance, a medical instrument could be critical to the well-being of a patient. On the other hand, a device to measure noise intensity is probably less critical.

To make an objective assessment of the level of criticality of usage, we need a scale which we give in increasing criticality as: **critical**, **business-critical**, **potentially life-critical** and **life-critical**.

One of the major problems in this area is that a *supplier* may well be unaware of the criticality of the application. The *user* may well assume that the measurement system is suitable for highly critical applications, while the supplier may well prefer to exclude such usage. For an example of this problem see **Buyer beware!** in Appendix A.4.

Legal requirements. Several measurement systems are used in contexts for which there are specific legal requirements, such as the WELMEC guide [WELMEC-2.3]. In such a context, a measurement system malfunction could have serious consequences.

To make an assessment, the *user* needs to know if there are any specific legal requirements for the measurement system and have a reference to these. (It may be necessary to check what current legislation applies.) For a general-purpose measurement system, the *supplier* may not have

produced the measurement system to satisfy the legal requirements of a particular usage.

Complexity of control. The control function of the software can range from being almost non-existent to having substantial complexity. Aspects of the control will be visible to the operator in those cases in which operator options are available. Some control may be invisible, such as a built-in test function to help detect any hardware malfunction.

Many aspects of control are to make the device simpler to use, and protect the operator against mis-use which might be feasible otherwise. This type of control is clearly highly advantageous, but it may be unclear if any error in its operating software could produce a false reading. Hence aspects of the complexity of the user-instrument interface are considered here.

Very simple. An example of a very simple control is when the measurement system detects if there is a specimen in place, either by means of a separate detector, or from the basic data measurement reading. The result of this detection is to produce a more helpful display read-out.

Simple. An example here might be temperature control which is undertaken so that temperature variation cannot affect the basic measurement data.

Modest. An example of modest complexity arises if the measurement system takes the operator through a number of stages, ensuring that each stage is satisfactorily complete before the next is started. This control can have an indirect effect upon the basic test/measurement data, or a software error could have a significant effect upon that data.

Complex[1]**.** An example of complex control is when the software contributes directly to the functionality of the measurement system. For instance, if the measurement system moves the specimen, and these movements are software controlled and have a direct bearing upon the measurement/test results.

The *supplier* should be able to provide the *user* with the measure of the complexity of control on the above scale.

Complexity of processing of data. In this context, we are concerned with the processing of the raw data from the basic instrument (i.e., the instrument without the software). In the case of software embedded within the measurement system itself, the raw data may not be externally visible. This clearly presents a problem for any independent assessment; however, it should be the case that the nature of the raw data is clear and that the form of processing is well-defined. Calibration during manufacture would typically allow for 'raw data' to be displayed in appropriate units. (Subsequent to the calibration during manufacture, the raw data may not be available to the user.)

[1]This term should not be confused with that in complex arithmetic — readers are invited to propose another term should they find it confusing.

Very simple. In this case, the processing is a linear transformation of the raw data only, and with no adjustable calibration taking place.

Simple. Simple non-linear correction terms can be applied here, together with the application of calibration data. A typical example is the application of a small quadratic correction term to a nearly linear instrument which is undertaken to obtain a higher accuracy of measurement.

Modest. Well-known published algorithms are applied to the raw data. The assumption here is that the algorithms used are numerically stable. For an example of a problem that can arise in this area, see **Numerical instability** in section A.2.

Complex. Anything else.

The *supplier* should be able to provide the *user* with the measure of the complexity of data processing on the above scale.

A risk assessment must be based upon the visible information. Some complex devices may internally record information which is difficult or impossible access. Examples of such information is the selection of operator options, or low-level data within a complex system. For programmable systems, it should be possible to reconstruct the program from a listing, and repeat the execution from the data recorded from a prior execution.

The *user* needs to study the information provided by the *supplier* to confirm that the risk assessment is based upon a correct understanding of the measurement system.

4.2 Some examples

In the case of Druck's pressure transducer (see 3.6 and [SMART]), from the supplier's point of view it is **business-critical**, but from the point of view of its application in the nuclear power plant, it is **life-critical**. In practice, this mis-match requires that any additional validation needed from safety application would need to be funded by the users.

In the cases of safety applications, additional legal requirements apply. However, the scope of this research does not currently cover the safety issues — here we merely point out the potential risks.

There is no doubt that the majority of measurement systems perform **very simple** or **simple** processing on the raw data. It would be reasonable to assume the risks of the software error were quite low, but the example of the vacuum pressure measurement device (see 3.7) shows that such risks cannot be totally ignored.

4.3 A warning from history

From 1985, for a period of 5 years, the FDA had been warning the pharmaceutical industry that it was expecting it not only to validate their processes but also the computer systems that were involved. Unfortunately the pharmaceutical industry did not take this warning on board.

In 1991 the FDA conducted a foreign inspection of pharmaceutical companies in Europe and, for the first time, included computer validation.

At a number of sites the FDA issued a 483 (major non compliance) against the computer control system of an Autoclave for not being validated. An autoclave is used to sterilise drugs after container filling.

The non compliances were noted as:

- No formal plans stating the validation requirements (i.e., Validation Plan).
- Specifications were not available to define the intended software operation.
- No qualification protocols defining testing or acceptance criteria.
- Inadequate accuracy checks, input-output checks and alarm testing.
- Inadequate change and version control of software.
- No final review of the evidence demonstrating validation (i.e., Validation Report).

A validation plan was immediately drawn up for the Autoclave and put into action. It covered specification generation, installation qualification (IQ — hardware testing), operational qualification (OQ — software testing), code walk-throughs and supplier audits.

The results were surprising.

1. In the IQ/OQ phases one alarm code was found to be missing, two limits were crossed over and the real time clock was found to be of insufficient accuracy over a period of time to support integration techniques.
2. The more critical areas of code were checked by walking through the program to ensure that it had been correctly coded to function in the way it was designed and that it had been written to the company defined standard.

   The Code Walk-through addressed:

   - Software Design.
   - Adherence to coding standards.
   - Software Logic
   - Absence of redundant code.
   - Critical Algorithms.
   - Software control.
   - Level of code comments/explanation.

   The software was well written but the main algorithm for calculating F_0 (F_0 is defined as the time during which sterilisation is actually performed at 121°C i.e., time of microbial destruction effectiveness) was found to ignore summation of a small area under the curve giving a lower F_0 calculation over the sterilisation period. The result of this was to subject the product to a longer sterilisation period than required.

   The computer manufacturer was informed of the findings and the company responded quickly in rectifying the problems.

3. In setting up the supplier audits for the Autoclave it was discovered that the computer system was built and tested by a computer manufacturer and then application programmed and fitted by the autoclave manufacturer. A decision was taken to audit both companies.

   The computer manufacturer was situated in a building where the complete hardware and software design process was carried out. The full life cycle was checked out, right through to testing and final shipment packing to Autoclave manufacturer. No problems were encountered and a high degree of confidence was recorded in the manufacture of the computer system.

   A visit was then paid to the Autoclave manufacturer and the audit trail started from where the tested computer system was delivered into stores. The stores was at one end of a very large workshop area where approximately 20 autoclaves of various sizes were in different stages of manufacture. The computer system was followed to a small computer workshop, next to the stores, where the correct sequence software was installed for the Autoclave it was to control. From there it was followed to the workshop floor where it was stored under the nearest workbench to the autoclave it was designated for. Building an autoclave, which is essentially a large kettle, is mainly an engineering and plumbing task hence there were many workbenches around for people to use. It was noticed that some of the unpacked, unprotected computer systems were beneath bench vices and in a number of cases metal filings had dropped onto the computer case which had open air vents in it. A number of other major problems were found, one being that there were no written tests or recorded test data. It depended on the person building the autoclave and the test plan he kept in his head.

   Once the problems were pointed out to the Autoclave manufacturer, the company responded very quickly and now have a completely different method of computer integration and test which ensures the computer controlled autoclave is produced to the highest quality.

4.4 Conclusions

The above example has been supplied to NPL by industry as an example of the problems actually encountered. The originators wish to remain anonymous.

The conclusions are clear: a risk assessment needs to be undertaken as early as feasible and the consequences followed through the entire process of the use of the system.

In the above case, the legal requirements were crucial. Even without that, the pharmaceutical supplier needs to be assured that the Autoclave performs the intended function, since a failure could have terrible consequences. In this case, the Autoclave supplier accepted the requirement for an audit and undertook the necessary remedial action. The purchase of off-the-shelf equipment may well present more problems in gaining visibility of the necessary information.

Chapter 5

Software Implications

At this stage, we are looking at the measurement system as a black box but assuming that some questions can be asked (and answered) which might not be directly apparent from the measurement system. The underlying reasoning behind the questions is to assess the affects of the risk factors involved. If some key questions can not be answered, then clearly any assessment must be incomplete.

5.1 Software issues

The first part is to go through the previous section of this Guide and characterise the measurement system in terms of all the parameters mentioned there.

We now have a set of additional issues to attempt to resolve as follows, in the form of open questions:

1. What degree of confidence can be obtained in the measurement system merely by performing 'end-to-end' tests[1], i.e, using the measurement system with specimens of known characteristics?

   Such tests just regard the entire measurement system as a black-box and effectively ignore that software is involved. To answer this leading question you need to take into account the risk factors noted above. For instance, if complex software is being used which uses unpublished algorithms, then high confidence cannot be established.

   As a *user*, you may have no visibility of the software and hence this could be the only form of testing that could be undertaken. This will clearly limit the assurance that can be obtained, at least if the software alone could cause the measurement system to malfunction.

   As a *supplier*, you may need to provide some additional information on the measurement system for those users who require high assurance.

   (Note that this type of testing is distinct from conventional black-box testing of the software since the software is only exercised in conjunction with the basic instrument.)

[1] The term 'end-to-end' is used for testing the measurement system as a whole — readers are invited to propose another term should they find this confusing.

2. In the case in which the processing of the basic data is **modest** or **complex**, can the raw data be extracted so that an independent check on the software can be applied?

   Consider a case in which a Coordinate Measuring Machine (CMM) is being used to determine the degree of departure of an artifact from a perfect sphere. The basic CMM will provide the x, y, z coordinates of points on the surface of the artifact, and additional software will determine the departure from the sphere. For high assurance, it is essential that the additional software should be tested in isolation, preferable using Reference Test Sets [ISO 10360-6].

3. Has essentially the same software for the data processing been applied to a similar measurement system for which reliability data is available? (Note that there is a degree of subjective judgement here which implies that the question should be considered by someone who is suitably qualified.)

   The *supplier* may regard such information as commercially sensitive and hence is not provided to the user, or is only available under a non-disclosure agreement.

4. For this measurement system, is there a record available of all software errors located? Under what terms, if any, can this information be made available?

   The *user* could reasonably request this information and the *supplier* should be able to supply it, perhaps under some non-disclosure agreement. Note that the existence of such a log is a requirement of quality management systems, like ISO 9001 [ISO 9001].

5. To what extent can the complexity in the control software result in false measurement/test data being produced?

   The *user* is likely to have great difficult in answering this question while the *supplier* may be confident of the measurement system, but finds it difficult to provide objective evidence to support that confidence.

6. If the operator interface is complex, can this be assessed against the documentation? How important is operator training in this?

   In this case the *user* is in an ideal position to independently check the documentation. The *supplier* would no doubt welcome constructive feedback.

5.2 Software Integrity Levels

At this point, the *user* should have sufficient information available to make an assessment of the integrity required of the software taking into account all the factors above including the target risk to be taken. This assessment should be as objective as possible, but is bound to have a modest degree of subjectivity. If answering the six questions above is straightforward, raising no problems, then the **Software Integrity Level** is the same as the complexity of the data processing above. Hence, unless answering the above questions reveals

additional problems, **very simple** complexity of data processing would have a **Software Integrity Level** of 1.

Thus classify the result of this process as follows:

Software Integrity Level 1. The data processing software is **very simple**. No significant problems were revealed in the analysis.

Software Integrity Level 2. The data processing software is **simple** or some problems were encountered and the processing was **very simple**.

Software Integrity Level 3. There is at least one major unquantifiable aspect to the software. This could be an inability to check the software since there is no facility to provide the raw data (combined with complex processing). Another possibility might be that the control software influences the basic measurement/test data in ways that cannot be quantified.

Software Integrity Level 4. Here we either have complex processing which is difficult to validate, or processing of modest complexity with significant additional problems (or both!).

As the software integrity level increases, so should the risk of errors be reduced due the application of more rigorous software engineering techniques, which is the topic of section 5.4.

Once the *user* has determined the software integrity level, this figure should be agreed, if possible, with the *supplier*.

5.3 Computing the software integrity level

It has been suggested that there should be an algorithm for computing the software integrity level from the information which is requested in the last two sections. Each key factor is on a 4-point scale, as is the resulting software integrity level. Hence one possibility is:

Software Integrity Level = max(Usage, Control, Processing) levels.

This suggestion has not been developed further since it seems to be difficult to take into account all the factors necessary. For instance, even the maximum function above is not quite correct since the complexity in the control function could be off-set by other factors.

On balance, it seems more appropriate for the factors to be determined, the check lists used, and then a subjective judgement made (which could well be based upon the formula above). The important aspect is to show how, and on what basis, the software integrity level was determined. Some alternative methods are given in [IEC 61508].

5.4 Software validation techniques

Given a **Software Integrity Level**, then the risks can be mitigated by the application of suitable software validation techniques. If the *supplier* can assure *the user* that appropriate techniques have been applied, then the use of the measurement system in the agreed role can be justified.

The most appropriate techniques to apply depend upon the nature of the software, and are considered in detail in part II.

As a *user*, the validation techniques used by the *supplier* may well be unknown. In that case, the Guide can be reviewed by the supplier who can be asked to comment, given the results of the risk assessment provided by the user.

A number of specific software techniques are described in part II, which could be developed to satisfy the guidance for general software testing given in [ISO 13233]. This would be a useful advance, since currently, methods like Reference Test Sets are only recommended in the specific context of Coordinate Measuring Machines [ISO 10360-6].

5.5 Software and quality management

For any software integrity level, basic practices must be established which could be a direct consequence of the application of ISO 9001 to software [ISO 9001, ISO 9000-3] or from the application of other standards. It can be claimed that ISO 9001 itself requires the application of appropriate (unspecified) software engineering techniques, especially for the higher levels of integrity. However, even ISO 9000-3 does not even mention many such techniques and hence we take the view that specific techniques should be recommended here, rather than depend upon the general requirements of ISO 9001. In the pharmaceutical sector, specific guidance has been produced which is effectively a version of ISO 9000-3 oriented to that sector [GAMP].

In the UK, it is reasonable to expect *suppliers* to be registered to ISO 9001, which in the case of software should imply the application of TickIT (although recently the TickIT Scheme has become a benchmark for any software certification scheme for ISO 9001, rather than a software specific extension to ISO 9001 registration; see [UKAS-TickIT]). If a supplier is *not* registered, then one lacks an independent audit on their quality system.

ISO 9001 provides a basic standard for quality management, whereas in practice, companies will continually improve their system if the aim is high quality. In any case, improvements in the light of experience are essentially a requirement of ISO 9001. The standard implies a defined life-cycle which is elaborated in [ISO 12207]. For those companies not formally registered to ISO 9001, it is necessary that a similar quality management approach is used and evidence for that can be produced.

To conclude, *users* should expect *suppliers* to be registered to ISO 9001, or to be able to demonstrate compliance to a similar level of quality management.

Chapter 6

Conclusions

The attempt to answer the two questions posed at the beginning of the Guide is limited by the information available. One must accept that the *user* of a measurement system may not be able to obtain information from the *supplier* to determine if appropriate software engineering practices have been used.

If end-to-end testing cannot be regarded as providing sufficient assurance of the measurement system, then a user must consult the supplier to obtain the assurance that is desired.

It must be accepted that assurance levels higher than the majority of the users of a measurement system require, can probably only be achieved by purchase of additional information from the supplier, perhaps even to the extent of developing a special-purpose measurement system.

Part II

Technical Application

Chapter 7

Understanding the Requirements

The main non-technical aspects of the requirements are covered in the first part of this Guide. The purpose of this part is to provide additional information of the software engineering aspects so that high assurance can be provided by a *supplier* in a form which a *user* can appreciate. Hence this part is directed more at the supplier than the user.

This Guide currently excludes safety aspects, specifically the certification of measurement systems to IEC 61508 [IEC 61508]. It is hoped that additional material can be added to cater for this in due course. We also exclude the more specialised area of Programmable Logic Controllers, since another guide is available covering this [IEE-PLC].

The general approach in Part I is followed here but with more detail. If high assurance is required, then the detail presented here is necessary. In essence, Part I only handles systems with a Software Integrity Level of 1.

The steps to achieve successful validation are illustrated in Figure 7.1. The following points should be noted:

1. If as a *user*, your assessment of the risk is higher than that envisaged by the *supplier*, then it may be difficult, or even impossible to obtain the assurance required.

2. As a user, you may be able to reduce the risks by choosing a simpler system with only just sufficient functionality to cover the application.

3. The Software Integrity Level is vital. Too low and the assurance is not obtained, too high and effort will be wasted.

4. The choice of validation methods will determine the cost.

5. In this Guide, we are not concerned with formal validation (such as would be required in many safety applications), so the report on validating a measurement system may be somewhat informal, as illustrated in Appendix B.

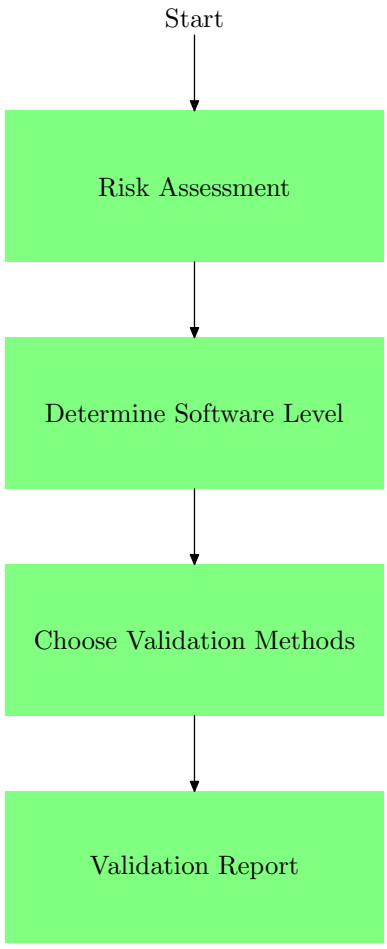

Figure 7.1: Steps in Validation

Chapter 8

Issues in Risk Assessment

Without a risk assessment one cannot be confident that any effort spent in validation will be worthwhile. Clearly, if one can be assured that the software within a measurement system can have no negative impact on the resulting measurements, then all that needed is to record the reasons why this conclusion has been reached. On the other hand, if the software is large and complex, and also performs an essential function in producing the output from the measurement system, and that output has a safety implication, then validation is essential.

The risk assessment should identify the key aspects so that the validation process can target these important aspects. Almost all complex software systems contain a high proportion of less critical code which could be checked by simpler and cheaper methods than the more critical parts.

Measurement is a numerical process. In consequence, one must be confident that the software processing of the numeric data produces a meaningful result. Techniques are available to ensure the numerical soundness of the computations within the measurement system, which should therefore be applied when the assurance they provide is required.

8.1 Risk factors

The fundamental problem for the *supplier* is that the criticality of the usage of a measurement system may be unknown, and therefore, a specific level must be chosen. If a *user* of a measurement system wishes to use it in a more critical application than the supplier has envisaged, then additional validation effort may be required, which can involve much analysis work.

The issues which need to be considered are collected together here as questions. Not all of these will be relevant for a specific measurement system.

1. For the *user*, what is the criticality of usage?
   critical / business-critical / potentially life-critical / life-critical
2. Is the *supplier* aware of the criticality of your application?
3. Are there specific legal requirements for the measurement system?
4. What are the consequences of a measurement system malfunction?

5. Is independent evidence needed of the software development process?
6. Does the measurement system require regulatory approval?
7. What is the complexity of control?
   very simple / simple / modest / complex
8. Does the measurement system perform built-in testing?
9. Do the control functions protect the operator from making specific errors?
10. Can an error in the control software cause an error in the basic test or measurement data?
11. Is the raw data available from the measurement system?
12. Does the measurement system contain local data, such as that derived from the last calibration?
13. Is the processing of the raw data strictly linear?
14. Is the processing of the raw data a simple non-linear correction?
15. Is the processing of the data restricted to published algorithms?
16. Have the algorithms in use been checked for numerical stability?
17. Would a numerical error, such as division by zero, be detected by the software, or would erroneous results be produced? This will typically depend upon the programming system used to produce the software, and can vary from no detection of such errors to elaborate indications of the exact point of failure. If no internal checks are applied, there is a greater risk of a programming error resulting in erroneous results.

One should not necessarily regard performing a risk assessment as a once-only function. Given an initial assessment, some validation work may be undertaken. As part of that validation, more information will be available which may imply that the risk assessment should be revised.

8.2 A simple example

A platinum resistance thermometer has a small amount of embedded software, mainly to provide a convenient interface for the user. The basic algorithm is to apply a small non-linear correction derived from the last calibration of the measurement system.

Assuming the application is not **life-critical**, and that we have assurance that the basic algorithm is as described above, then it would be reasonable to conclude that end-to-end testing of the measurement system as a whole will provide the assurance needed. In consequence, no specific software validation process is required. The assurance provided by the *supplier*, working under an appropriate quality management environment, should be sufficient.

For another, full example in which end-to-end testing provides high assurance, see page 66.

8.3 A complex example

Consider the example of the Autoclave in section 4.3. At first sight, the pharmaceutical company had every right to trust the Autoclave manufacturer to satisfy the requirements. However, FDA required visibility of the evidence which should support the correct operation of the Autoclave.

In undertaking the retrospective validation, errors were found in the computation of the (agreed) formula. Fortunately, the error was on the side of safety, but it seems clear that the error could have been unsafe so that the Autoclave did not sterilise the artifacts.

In terms of this Guide, the computation was **modest** since it involved integration over time. In this case, the errors in the clock compounded the problem.

The measurement of time and temperature to the accuracy required by this application is not demanding. In consequence, the trust placed by the pharmaceutical company seems reasonable. The observed errors in this case are therefore disturbing, especially since the correct operation of an Autoclave is almost certainly a safety factor.

The conclusion must be that even in cases in which the *supplier* could reasonably be expected to satisfy the requirements, *the user should ask for documentary evidence of this.*

Chapter 9

Software Integrity Requirements

The starting point here is that the software development process being used should have a rigour to match the software integrity level. This is the approach taken in several safety-critical software standards [IEC 61508, DO-178B].

Theoretically, it is possible to undertake the testing of software to establish the actual reliability of the software. However, there are strict limits to what can be achieved in this area [Lit 92], and hence the approach taken here is the conventional one of examining the software development process. In practical terms, software testing is expensive, and hence the most cost-effective solution uses other methods in addition to gain confidence in the software.

9.1 Software and quality management

Following the outline in Part I (see page 26), we assume that either ISO 9001 or a similar quality management system is being applied by the *supplier*. The application of such a quality management system to software should imply that a number of technical issues have been addressed and documented and that the following requirements are met:

1. There should be documents demonstrating that a number of issues have been covered such as: design, test planning, acceptance, etc. The acceptance testing should ensure that the operator interaction issues have been handled and validated.

2. There should be a detailed functional specification. Such a specification should be sufficient to undertake the coding. This level of information is typically confidential to the developer.

3. There should be a fault reporting mechanism supported by appropriate means of repairing bugs in the software.

4. The software should be under configuration control [NIS37]. This implies that either the software should itself include the version number, or the version can be derived from other information, such as the serial number

Software Integrity Level	*Recommended techniques*	*Reference*
2	Software inspection	10.1
	Code review	10.2
	Structural testing	10.3
	System testing	10.6
	Mathematical specification	10.8
3	Equivalence partition testing	10.3
	Regression testing	10.4
	Numerical stability	10.7
	Independent audit	10.10
	Stress testing	10.11
	Reference test sets	10.13
4	Statement testing	10.3
	Accredited testing	10.5
	Formal specification	10.9
	Static analysis	10.12
	Back-to-back testing	10.14

Table 9.1: Recommended techniques

of the device. In the case of free-standing software, it should be possible for users to determine the version number.

We assume that these requirements are met, whatever software integrity level is to be addressed by the software.

9.2 Recommended techniques

For each **Software Integrity Level**, a recommendation for level n also applies to any higher level. In Table 9.1, we list the recommended techniques, which are all defined in Chapter 10. Note that Statement testing, Equivalence partition testing, and Structural testing are all (software) component test methods.

Ensuring that numerically sound results are produced is clearly vital. In consequence, in all but the simplest of calculations, one must gain assurance by using only stable algorithms (i.e., ensuring the design is correct, see 10.7), or by checking the implementation using reference test sets (see 10.13), or by testing the implementation against a reference implementation (using back-to-back testing, see 10.14). (Very high assurance would require more than one technique.) Since we know of no other appropriate methods of gaining assurance for numerical calculations, one of these techniques should be used.

The fact that a technique is recommended at a specific level does not (in general) imply that not applying the method would imply poor practice or that all the methods should be applied. For instance, the example given under Accredited testing (see section 10.5), is a good choice precisely because other strong methods are not effective. Any design should involve a trade-off between the various relevant methods.

The issues here are again listed as questions. The first set are generic to all levels, and the others are specific to a given level.

1. What information is available from the measurement system supplier or developer of the software?
2. What confidence can be gained in the software by end-to-end tests on the measurement system?
3. Can the raw data be extracted from the measurement system?
4. Can the raw data be processed independently from the measurement system to give an independent check on the software?
5. Are software reliability figures available for a measurement system using similar software (i.e., produced by the same supplier)?
6. Is a log available of all software errors? Has this log been inspected for serious errors?
7. Does the control function have a direct effect on the basic test or measurement data?
8. Has an assessment been made of the control software against the operating manual? If so, by whom?
9. Do operators of the measurement system require formal training?
10. Have all the answers to the questions above in this list been taken into account in determining the software integrity level?
11. Has a list been made of all the unquantifiable aspects of the software?

9.3 Software development practice

The following lists increase in complexity with increasing in the **Software Integrity Level**. Since the requirements at level *n* imply those at level *n-1*, all the questions should be asked up to the level required.

9.3.1 Software Integrity Level 1

1. Is there design documentation?
2. Is there evidence of test planning?
3. Is there a test acceptance procedure?
4. Is there an error log?
5. What evidence is there of clearance of errors?
6. Is there a detailed functional specification?
7. Are security, usability and performance aspects covered in the specification?
8. How is configuration control managed?
9. Is there a defined life-cycle?

10. How can the user determine the version number of the software?
11. Have all changes to: hardware platform, operating system, compiler, and added functionality been checked?
12. Have all corrections been checked according to the defined procedures?
13. Have all the staff the necessary skills, and are these documented?

9.3.2 Software Integrity Level 2

1. Has software inspection been used on the project? If so, to what documents has it been applied and what was the estimated remaining fault rate?
2. What alternatives have been used if software inspection was not applied?
3. Has a mathematical specification been produced of the main algorithms used for processing the test/measurement data?
4. Is the processing code derived directly from the mathematical specification?
5. What form of structural testing has been applied, and what metrics of the level of testing have been produced?
6. What level of testing has been applied to the control and processing components of the software?
7. How has the completeness of the system testing been assessed?
8. Has the system testing covered all reasonable misuses of the measurement system?
9. What records are available on the system tests?
10. Are the security features consistent with any regulations or intended use?
11. Are the test strategies, cases, and test completion criteria sufficient to determine that the software meets its requirements?

9.3.3 Software Integrity Level 3

1. Has regression testing been applied? If so, at what point in the development did it start?
2. For what components has equivalence partition testing been applied? Has the technique been applied to the components processing the basic test or measurement data?
3. Has an independent audit been undertaken? Have all problems identified been resolved? Did the audit apply to the basic quality management system or to the software techniques as well?

4. Has the numerical stability of the main measurement/data processing routines been checked? Has a floating point error analysis been taken into account in formulating the mathematical specification of the routines?

5. Has stress testing been applied to the software? To what extent have the limits of the software been assessed by this testing? Has the stress testing revealed weaknesses in the system testing?

6. Have activities been undertaken in the development which are not auditable (say, no written records)?

7. Are known remaining software bugs documented? Are they adequately communicated to the user?

8. What methods have been applied to avoid structural decay? (See Appendix A.3.)

9.3.4 Software Integrity Level 4

1. To what components has statement testing been applied? What coverage was obtained?

2. Has a Formal Specification been produced of any of the software components? Has this revealed weaknesses in the functional specification, testing, etc.? Is it possible to derive an executable prototype from this specification to validate the equivalence partition testing?

3. What forms of static analysis have been undertaken?

4. Does accredited testing have a role in gaining confidence in the software? If a test suite is used for accredited testing, have all the results of other forms of testing been fed into this?

5. Is beta site testing undertaken?

6. Is memory utilisation testing undertaken, or can it be shown that such testing is not needed?

Chapter 10

Software Validation Techniques

Here we list specific recommended techniques. These are either defined here, or an appropriate reference given. A good general reference to software engineering is [SERB].

10.1 Software inspection

This technique is a formal process of reviewing the development of an output document from an input document. It is sometimes referred to as Fagan inspection. An input document could be the functional specification of a software component, and the output document the coding. An excellent book giving details of the method and its practical application is given in [INSPECT].

The method is not universally applied (in industry), but many organisations apply it with great success. It tends to be applied if the organisation has accepted it and endorses its benefits.

Scope

The requirements are an input document, an output document and terms of reference for the inspection. As such, the method can be applied to a very wide range of documents, and hence the processes producing such documents.

However, the method is only effective if used on a regular basis within an organisation. People need some training in the technique in addition to reading the reference given (or similar material).

Cost/Benefits

The process requires at least three people in an inspection. Experience shows that one needs to allow at least half an hour per page, and that some preparation time and time to write up the results of the inspection. Hence the cost is probably twice that of a single person undertaking a review — the alternative which is probably more widely used in industry.

The advantage is that after a successful inspection, one has more confidence in the output document than is attained by an individual review. Specifically, the result produces a document that is very likely to have consensus support within the organisation.

Conclusions

Organisations should consider applying the technique on a regular basis if it is felt that this depth of review of documents is advantageous.

For those organisations who already apply the technique, it is recommended that they review the proportion of critical documents that are reviewed. Internal quality procedures and records should indicate the areas where the main gains are to be found.

10.2 Code review

This technique is similar to software inspection, but carried out only on the source code of an application. The concept is to ensure the readability and maintainable of the source code, and potentially to find bugs. Ensuring the tracability of the code from the specification would be covered in a software inspection, but is not usually undertaken in a code review. We are considering the code review as a manual process, rather than the analysis of source text by tools as in static analysis (see page 53).

Organisations developing software typically have a procedure for undertaking code review which may list a number of aspects to be checked, such as ensuring that the programming language identifiers are meaningful.

If the developer makes the source text available to customers, then customers may undertake an independent code review.

Example

An extensive review of software source code for a robot measuring system by a user. In this review the following points were found.

In two modules the comment 'this needs checking' was present. Either the developer had not returned to check the code or had checked it and not removed the comment.

In one module a decision element was `if 1 = 1 then`. Over a page of software from the `else` option could never be entered.

In another module the following series of decision elements were found:

```
if a = 1 then A
```
was followed by
```
        else if a = 1 or b = 1 or c = 1 or d = 1 then B
```
which was followed by
```
        else if a = 1 or b = 1 or c = 1 or d = 1 then C
```
and again followed by
```
        else if a = 1 or b = 1 or c = 1 or d = 1 then D
```

Again elements `C` and `D` could never be entered.

Poor constructs led to the decision to do a 100% review of the code. This resulted in a number of actual errors being detected, e.g. the setting of an incorrect flag or warning lamp which would lead to the operator searching for a fault that had not occurred rather than the actual fault.

The developer had to perform a number of changes to the code.

Scope

Code review should be undertaken by any organisation developing software. In this Guide, we assume that either this technique is applied, or another method is applied of a similar strength to gain confidence in the software in its source text form.

Cost/Benefits

As a manual process, its success depends upon the skill of the reviewers and the level of effort applied. The costs can vary from a few minutes per page to over an hour per page if the intention is to study the source in depth. The varying costs is also reflected in the benefits which can be just an increased confidence in the readability, to a clear indication that the reviewer believes that the program is correct.

Conclusions

As part of software development, the process of code review should be undertaken or a process which is clearly superior, like software inspection.

10.3 Component testing

This is a basic software engineering technique which can (and should) be quantified. The software component to which the method is applied is the smallest item of software with a separate specification (sometimes called a module). It is very rare for the technique *not* to be applicable for a software development. The best standard, which is now available through BSI, is the British Computer Society standard [BS 7925]. The method is objective, repeatable and reproducible, and hence satisfies the requirements for accredited testing.

The BS 7925 standard allows for many levels of testing and in this Guide we select four levels as follows:

Structural testing. Several forms of structural testing are defined in the standard, but not to any specified level. In this context, we specify branch testing with 50% coverage.

Equivalence partition testing. It is recommended to undertake this to 100% coverage. This is complete functional testing at the component level. This is to be applied to those components handling the basic measurement/test data.

Statement testing. It is recommended to undertake this to 100% statement coverage for those components handling the basic measurement/test data.

If a statement has not been executed, then a reason for this should be documented. Defensive programming techniques and detection of hardware malfunction give rise to statements that cannot be executed (but are quite acceptable).

Boundary value testing. In this case, which can be seen as an addition to equivalence partition testing, values are chosen which lie on the boundary between partitions. This form of testing is designed to show that the boundary cases themselves are correctly handled.

Example

In the [SMART] research project, it was concluded that equivalence partition testing and boundary value testing are the most cost-effective methods in this area.

Scope

The technique is very general and can be applied to any software written in a conventional programming language. The method is widely used and understood. Metrics are available which provide a measure of the thoroughness of the actual testing undertaken.

The fundamental drawback of the method is that 'missing code' cannot be identified. For instance, if a routine is defective in not protecting itself against invalid input, then structural coverage testing will not detect this.

The method cannot be applied to software only available in compiled form, such as library routine provided by other parties (i.e., COTS).

The technique should be applied to each component with the software. However, it may be difficult to produce test cases for some components in isolation from the rest of the software. When a component is not isolated, it may be difficult to produce all the test cases that are needed to attain a high coverage figure.

In the context of measurement system software, it may be difficult to apply if the software cannot be isolated from the measurement system itself, since it may not be possible to get complete repeatability. (This is when the entire software is regarded as a component.)

Cost/Benefits

The only cost-effective method of applying the technique is with the aid of tools which automatically indicate which statement, etc have been covered in the testing. Such tools are used on a routine basis by some software development teams but not others.

Cost can be high if, say, 100% coverage is required. Under such circumstances, work in analysis of the algorithm to produce the necessary test cases can be very demanding.

Equivalence partition testing is effectively black-box testing for a component done in an ideal manner. This method should be applied for the most critical software components of a system. The method is simple and easy to apply to simple software.

Conclusions

Software development teams should ensure they have the necessary tools to perform at least statement coverage analysis without a significant burden on their development process. The author of a component should review that statement coverage data to see if crucial test cases have been omitted.

Initial test cases should be derived from equivalence partition testing. The statement coverage data then shows if the testing is adequate.

10.4 Regression testing

This technique requires that tests are developed and used to re-test the software whenever a change is made. Typically, sometime before the first release, a set of tests will be designed and run on the software. From that point on, all errors located should result in an addition to the set of tests of an example which would detect the bug.

To be effective, one needs a method of re-running the set of tests automatically. The technique is very good for software that is widely used and for which initial bugs are not a major problem. The effect of the method is that subsequent releases of software should be very reliable on the unextended facilities.

Scope

This method is very powerful for software that is being constantly updated and enhanced. Correspondingly, it is not effective for software embedded within a measurement system (say) which has to be without a serious defect on the first release. If a field upgrade is needed with such software, there could be very serious cost implications.

To be applied, the software must be separated from the hardware of the measurement system.

Cost/Benefits

Individual test cases need to be recorded within a system which allows for minimal effort to re-execute them. Hence, give a modest additional effort in the initial tests, further releases of the software should be tests very effectively.

(One reason why compilers, which are typically very complex, are actually very reliable, is that almost all developers use regression testing.)

Conclusions

Management should decide, on the basis of the type of software, if regression testing should be undertaken.

10.5 Accredited software testing using a validation suite

This technique requires that a set of tests be developed (the validation suite) against which the software can be tested. This is appropriate for software having an agreed detailed specification, such as compilers and communication software. Accredited testing ensures that the tests are run and the result interpreted correctly, with the specific requirements of objectivity, repeatability and reproducibility.

The method is significantly stronger if the set of tests is updated regularly by means of regression testing. This implies that errors in any implementation will result in tests being applied to all (validated) systems.

This form of testing provides an ideal basis for certification.

Example

An example of this form of testing for a measurement system is that being proposed in the area of Nuclear Medicine [COST B2]. Here, gamma-camera pictures are taken of patients when being treated with substances containing radio-active trace elements. The camera output is translated into a standard file format, but the difficult numerical part is the analysis of the picture to give the basic information for a medical diagnosis. Other strong methods, such as a mathematical specification cannot be applied, and hence this method provides a means for the whole international Nuclear Medicine community to gain confidence in analysis software.

Note that the application is **potentially life-critical** and the complexity of the processing of data is **complex** which implies a high **software integrity level**, say 3. At this level, the technique of accredited testing is not actually recommended (see Table 9.1 on section 9.1), but it is one of the few methods which can provide reasonable assurance in this context.

This method is made more effective by means of software phantoms which are pictures for whom an agreed diagnosis is available (as least in the cardiac and renal areas), as explained in [COST B2].

Scope

All the applications of this method are to generic application, since as compilers, for which many instances are available (to effectively the same specification).

A potential example in the area of length metrology would be tests for the software that analyses data from Coordinate Measuring Machines.

For such a method to be effective, there has to be an underlying test strategy which can be applied to a range of software products. For instance, it is unclear that the method could be applied to word processors, since there is no generic specification (with sufficient precision for testing).

On the other hand, the method *could* be applied to the basic mathematical operations within a spreadsheet, and this issue is being explored as part of the SS*f*M programme.

Cost/Benefits

The costs to produce the generic test data can be quite high so that the method is limited to those cases in which there is a national or international consensus that such a development should be undertaken.

Conclusions

In conjunction with industry, the SS*f*M programme should formulate proposals for the application of this technique where high assurance is needed for generic products. This will be considered for a future SS*f*M programme.

10.6 System-level software functional testing

This technique is based upon testing the entire software as a black box by a careful examination of the functionality specified and ensuring that every aspect of the functionality is tested. An ISO standard is based upon application of this test method [ISO 12119].

The method can clearly be applied to any system. In this context, we are considering its application to the complete software, but separate from the measurement system hardware. This limits the application to those cases in which such separation is possible.

The problem with the technique is the cost of constructing the necessary test cases, and ensuring that the test cases are sufficient. Since errors have been reported in many software systems which have been very carefully tested, one cannot gain very high assurance from this method.

However, in cases in which the software has a simple structure, such testing may be sufficient.

Example

NPL has recently tested a software product to [ISO 12119]. The product was a small item of software to detect if a PC had a problem with the clock in the millennium change, and correct it, if required.

The program came on a floppy disc, with essentially no options. Hence the testing was mainly to ensure that different types of PC would have their clocks corrected, as claimed in the supplier's specification. This could be undertaken by first seeing if the PC had a problem, then running the software, and finally checking the machine again.

The testing was undertaken in line with UKAS requirements, and the test report produced is available for inspection [Y2000].

If the program had 20 different installation options, the the complete testing required by the standard could require 2^{20} test cases and hence be too costly to undertake.

Scope

The technique can be applied to software systems which are sufficiently simple that complete functional testing is economically viable.

Unfortunately, the method is not viable for most complex software.

Cost/Benefits

As noted above the problem is that the costs can be high unless the software in very simple (at least in its specification). If high assurance is required and the claims made for the software can be converted to appropriate test cases, then the method can be applied.

Conclusions

The method should be considered when high assurance is required by independent test, and the claims made for the software lends itself to a practical number of tests.

10.7 Numerical stability

It is unreasonable to expect even software of the highest quality to deliver results to the full accuracy indicated by the computational precision. This would only in general be possible for (some) problems that are perfectly conditioned, i.e., problems for which a small change in the data makes a comparably small change in the results. Problems regularly arise in which the conditioning is significant and for which no algorithm, however good, can provide results to the accuracy obtainable for well-conditioned problems. A good algorithm, i.e., one that is numerically stable, can be expected to provide results at or within the limitations of the conditioning of the problem. A poor algorithm can exacerbate the effects of natural ill-conditioning, with the consequence that the results are poorer than those for a good algorithm.

Software used in scientific disciplines can be unreliable because it implements numerical algorithms that are unstable or not robust. Some of the reasons for such failings are

1. failure to scale, translate, normalise or otherwise transform the input data appropriately before solution (and to perform the inverse operation if necessary following solution),
2. the use of an unstable parametrisation of the problem,
3. the use of a solution process that exacerbates the inherent (natural) ill-conditioning of the problem,
4. a poor choice of formula from a set of mathematically (but not numerically) equivalent forms.

The development of algorithms that are numerically stable is a difficult task, and one that should be undertaken with guidance from a numerical analyst or someone with suitable training and experience. It requires that the intended data processing is posed sensibly and, if 'off-the-shelf' software modules are used, that such software is appropriately.

There are established high-quality libraries of numerical software that have been developed over many man-years and cover a wide range of computational problems. Examples include the NAG library [NAG] (which is available in a number of computer languages and for a variety of platforms), LINPACK

[LINPACK], and NPL libraries for data approximation [NPL-LIB] and numerical optimisation.

Example

An example which shows numerical (in-)stability is the calculation of the angle (θ) between two unit vectors. Given two vectors, $\mathbf{a}$ and $\mathbf{b}$, each of unit length, two mathematically equivalent formulae for evaluating the angle are

$$\theta = \cos^{-1}(\mathbf{a}^{\mathrm{T}}\mathbf{b})$$

and

$$\theta = 2\sin^{-1}\left(\frac{1}{2}||\mathbf{b}-\mathbf{a}||\right)$$

For vectors that are very nearly parallel, i.e., $\mathbf{a}^{\mathrm{T}}\mathbf{b} = 1-\delta$, where $|\delta|$ is much less than 1, the first formula gives $\theta = \sqrt{2\delta}$ (using the approximation $\cos\theta = 1 - \frac{\theta^2}{2}$).

The smallest θ computable in this way is the square root of the computational precision, i.e., approximately 10^{-8} radians for IEEE arithmetic. Thus, the first formula is unable to detect an angle smaller than 10^{-8} radians unless it is computed as zero, whereas the alternative formula can be expected to return accurate values.

This example has serious consequences for those concerned with implementing the procedures described in the ISO Draft International Standard [ISO 10360-6]. This Standard is concerned with testing software for computing Gaussian best-fit geometric elements to measured data that is used in industrial inspection and geometric tolerancing applications. The Standard requires that the unit direction vector used in the parametrisation of such geometric elements as planes, cylinders and cones and returned by test software is compared with a reference solution by computing the angle between the test and reference vectors. The Standard defines acceptance of the test vector if this angle is smaller than 10^{-8} radians. It is clear from the above analysis that if the first formula is used to evaluate the angle, this acceptance criterion can never be satisfied no matter how close are the test and reference vectors unless the angle is computed as zero. On the other hand, there is no problem with undertaking the comparison if the second formula is used.

Scope

Numerical stability is an issue for mathematical software which does all but the simplest arithmetic.

Cost/Benefit

There are costs in finding a sufficiently stable numerical algorithm for a particular application and there may be increased costs of development and computation in using a numerically stable algorithm as compared to the "obvious" calculation. However the costs from inaccuracy in the output from using an unstable algorithm can be catastrophic; they can certainly lead to "out of bounds" errors (e.g. 1/0, or $\sqrt{x}$ where $x < 0$) in subsequent calculations.

Conclusions

It is usually right to use a numerically stable algorithm if one is known. The use of respected software libraries for generic calculations will take advantage of the numerically stable algorithms used by such software. Otherwise, it is usually right to do some investigation of the stability of algorithm being used. If a numerically stable algorithm is much more expensive to implement or use, then an analysis of the benefits in terms of the desired accuracy and robustness is needed.

10.8 Mathematical specification

Such a specification gives the output data values as a function of the input data values. This method is suitable for the simpler processing of basic measurement data, and should clearly be expected. The mathematical function may well not be the way the actual output is computed, for instance, the specification may use the inverse of a matrix, while the results are actually computed by Gaussian elimination. This method should avoid a common error of not specifying the exact effect of the end of a range of values. It is not easy to apply the method to digital images (say), since the algorithms applied are quite complex so that any 'complete' specification is likely to be very similar to the software itself.

The mathematical specification needs to be validated against the underlying physics. This includes establishing that the model describes the system sufficiently well and ensuring that the errors introduced by the system are fully understood.

In some cases, the mathematical specification could be based upon an empirical relationship rather directly upon the underlying physics. The validation of this empirical relationship is covered in the Modelling Theme of the SS*f*M programme.

Example

Almost universal, see [SMART] for an example of a pressure transducer. In that case, the mathematical specification has been analysed to show the accuracy of the computation in the implementation.

Scope

> *I often say that when you can measure what you are speaking about and express it in numbers you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of* science*, whatever the matter may be.*
>
> Lord Kelvin, Lecture on Electrical Units of Measurement, 3rd May 1883.

The above quotation expresses the universal nature of mathematical specifications in the area of measurement.

Cost/Benefits

Given its universal nature, the only question is the rigour with which the method is applied. For linear systems, supported by calibration, perhaps little needs to be done.

Conclusions

Apply in all cases in which the mathematical formulae are not obvious.

10.9 Formal specification

Several methods are available for providing a specification in a completely formal way which can handle most functional aspects of a specification. The best known methods are VDM [VDM-SL] and Z [Z]. For the author's personal views of this method, see [FM].

Example

The use of such techniques is more frequently associated with safety and security systems, rather than metrology. However, an example of such a specification involving measurement is from a system to alleviated the effects of gusts on an aircraft [Monitor].

In this application, three basic measurements are combined and then, if the alleviation mechanism is invoked, the Flight Control System (FSC) must be informed to take the necessary action. The key issue here is that the FSC is **life-critical**, and therefore, by association so is the gust alleviation system. Moreover, the logic is complex enough to demand very careful specification which in this case, is achieved by the application of the Z language.

Scope

Formal specification languages have wide applicability, but are most effective when used to specify complex discrete systems. If all the relevant parties can read such a specification, it substantially reduces the risk of an ambiguity resulting in an error in the software.

Cost/Benefits

Effective use requires people trained in the use of the technique being employed (such as VDM or Z). This could mean additional costs for a project which implies using Formal Methods only on the most critical systems.

Conclusions

If a system has a large number of discrete states, all or most of which could influence the measurement results, then it is unlikely that sufficient assurance can be gained from testing alone. In such cases, the use of a Formal Specification should be considered.

10.10 Independent audit

In the UK, independent audit to [ISO 9001] is widely established. This provides evidence to third parties of a **software integrity level** of 1. It would not require that the stronger (and more expensive) techniques are applied, nor that the recommendations here are applied. In consequence, auditing to comply with the other standards mentioned in Chapter 2 would be better.

Example

NPL, and many other organisations, are independently audited to ISO 9001. In the context of measurement (and testing), assessment by UKAS to [M10] is probably more relevant.

Scope

Universal, but such auditing/assessment is not as widespread in the USA as in many other developed countries. This implies that assurance via independent audit may not be effective for some imported measurement system.

Cost/Benefits

The cost of auditing and assessment depends largely on the number of man-days of auditing effort. The benefits are in a similar proportion.

Conclusions

If you have any doubts about the standing of a measurement system supplier, you should ask about registration to [ISO 9001]. If the supplier calibrates any measurement system, including their own, then they should be assessed independently to [ISO 25] or an equivalent standard. As a user, you may be able to calibrate the measurement system yourself, but nevertheless, the supplier should be able to demonstrate this capability too.

10.11 Stress testing

This testing technique involves producing test cases which are more complex and demanding than are likely to arise in practice. It has been applied to testing compilers and other complex software with good results. The best results are obtained when the results can be automatically analysed.

For a paper on this method, see [STRESS].

Example

Consider a measurement system which claims to be able to make measurements even when there is some background 'noise' on the input data. Then a method of stress-testing would be to take a set of input data without noise, and then add noise. The correct answer is therefore known and if the noise can be added in an effective manner, then a stress-testing method can be applied. Clearly,

when the level of the noise gets too high, the measurement system cannot be expected to give a reading. However, a false reading should not be given.

The generation of the 'noise' by software should be easy to undertake, and since the correct result is known, the automatic testing of the measurement system should be possible with perhaps hundreds of individual test cases. If any failures arise, these can be examined manually to determine the source of the problem.

A special case of this technique is the application to video images. The starting image could be a high quality image or one produced artificially. An increasing level of noise is added to the image until the processing software fails. The final image which the software failed to process is then printed and a manual inspection made to see if a failure is acceptable in this case. It appears that this technique is widely used in validating software which processes video images.

See [STRESS] for an example testing compilers.

Scope

The method can be applied without knowledge of the internal structure of the software. It is particularly appropriate for complex software when it is possible to generate test cases automatically.

Cost/Benefits

The costs depend on the development needed for the test case generator. The benefits depend upon the assurance that passing such tests gives. For the example using compilers [STRESS], the cost of producing the test generator was quite high. However, the cost may be acceptable because very few other means are available for independently generating many, complex test cases.

The technique applied to video images does not require a test generator, since noise can be added to captured images. Also, graphics processing packages can be used to manipulate images prior to the addition of the noise. However, very high confidence in image processing is hard to achieve since the variety of images that can be produced is very large.

Conclusions

Consider this method for complex software which can only be tested as a black-box and for which high assurance is required. Given these requirements, then a cost/benefit analysis should be undertaken to see if the method is viable.

10.12 Static analysis/predictable execution

This technique determines properties of the software primarily without execution. One specific property is of key interest: to show that all possible executions are predictable, i.e, determined from the semantics of the high level programming language in use. Often, software tools are used to assist in the analysis, typically using the programming language source text as input. In general, the analysis techniques employed can be very minor (say, all variables

are explicitly declared), or very strong (formal proof of correctness), but the goal of showing predictable execution should be cost-effective for high integrity software.

For a general discussion on static analysis, see [SA-Survey].

Example

An example of this technique is to be found in the [SMART] project. NPL analysed the C code within Druck's pressure transducer using the NPL C code validation service tools.

The code was first converted to work in a free-standing manner on a Linux system. Then the code was compiled with a special compiler which warns of any non-standard (or potentially non-standard) constructs. Lastly, the code was executed with test cases designed to ensure statement coverage and boundary value testing. A few problems were located which were either corrected, or shown not be of concern in this application.

This form of testing is the very comprehensive method of demonstrating the correctness of the source code of an application. It will not locate errors in the specification of the software, nor errors which an incorrect compiler could introduce.

Scope

All software written in languages with precisely defined semantics.

Cost/Benefits

Effective use depends upon appropriate tools. Such a tool exist for C. For Ada and Java, most of the checking is undertaken by the compiler. No tool is available for C++, nor for most proprietary languages.

Conclusions

Consider the application of the method when the source text is available and high assurance is required.

10.13 Reference test sets

There is a growing need to ensure that software used by scientists is fit for purpose and especially that the results it produces are correct, to within a prescribed accuracy, for the problems purportedly solved. Methodologies, such as that presented in [Ref-sets], have been developed to this end. The basis of the approach is the design and use of reference data sets and corresponding reference results to undertake black-box testing.

The approach allows for reference data sets and results to be generated in a manner that is consistent with the functional specification of the problem addressed by the software. In addition, data sets corresponding to problems with various 'degrees of difficulty' or condition (section 10.7), and with application-specific properties, may be produced. The comparison of the test and reference

results is made objective by the use of quality metrics. The results of the comparison are then used to assess the degree of correctness of the algorithm, i.e., the quality of the underlying mathematical procedure and its implementation, as well as its fitness-for-purpose in the user's application.

The methodology has been applied successfully in particular areas of metrology. In dimensional metrology, for example, coordinate measuring machines (CMMs) are typically provided with software for least-squares (Gaussian) geometric element fitting. The methodology provides the basis of an ISO Standard [ISO 10360-6] for testing such software, and it is intended to base a testing service on this Standard. Data sets have been developed in such a way that the corresponding reference results are known a priori. Consequently, there is no reliance on reference implementations of software to solve the computational problems, but the generation of the data sets is dependent on a set of simpler 'core' numerical tasks that are well understood.

Example

In a number of metrology areas it is necessary to identify *peaks* within spectral and other traces and, particularly, to quantify a peak in terms of parameters such as location, amplitude and width. Algorithms for peak analysis occur in a range of measurement-system software [Chem, Dyson, AOAC]. Commonly, a particular functional form for the mathematical representation of the peak is assumed, e.g., Gaussian, Lorenzian or modified Gaussian, this form fitted by, e.g., least squares to numerical values from the trace, and the required parameters so determined. A widely-used form is the Gaussian model $y = A\exp((-(x-\overline{x})/(2\sigma^2)))$, where $\overline{x}$ represents location, A amplitude and σ peak width.

In order to determine whether peak-fitting software is functioning correctly, reference test sets can be devised. Such test sets consist of synthesised reference data sets and corresponding reference results [R-26]. Such data sets can be generated corresponding to ranges of key variables (extent of peak spanned by the data, peak height) width ratio, signal/noise value, etc.) using the null-space data generation technique [Design, R-25, Ref-sets, Grade-sets]. These ranges should be selected to correspond to those expected to be experienced in the use of the measurement system.

Scope

The null-space method of data generation is very general, being applicable to a wide range of fitting and optimisation problems [Opt, Opt-test]. Function forms other than Gaussian peaks can be considered, and fitting measures other than least squares can be entertained. The use of the reference data sets permits an assessment of the extent to which a numerically-satisfactory algorithm solution has been implemented. For instance, it will be possible to deduce whether a potentially-unstable process such as one based on the formation of the normal equations [R-27, Golub], as opposed to the direct use of the observations (design) matrix, has been employed. Moreover, it can be discerned whether conceptual errors in the solution process have been made, such as fitting the logarithm of the Gaussian model (which is equivalent to a quadratic polynomial) to the logarithm of the data. Within such approximate solutions processes there

are "degrees of correctness" such as partial compensation, through the use of appropriate weighting functions, for such model and data transformations, or the incorporation of such functions. Again the use of reference data sets can help to identify such techniques, the adequacy of which depends on practical quantities such as the signal-to-noise ratio.

Cost/Benefits

The technology for generating data sets as above is established. Its implementation in any particular instance may be governed by the ease with which the measurement system manufacturer has made provisions for software testing using simulated data. Since such testing should be fundamental to measurement system development, these would be an advantage in the manufacturer providing comparable facilities for user testing to establish fitness for purpose.

Conclusions

Synthesised reference test sets provide a powerful facility for testing the degree of numerical correctness of measurement software. Techniques such as the null-space method can be used to generate test sets with known solutions. It is only necessary to be able to provide a characterisation of the solution to the problem purportedly solved by the software. This characterisation is closely related to the functional specification of the software, which has to be prepared in any case.

It is necessary that the measurement system manufacturer provides a facility by which data sets may be input to and corresponding results output from the software. Such provisions should be a notional part of the manufacturer's system configuration and testing regime.

Arguably the most important feature of reference test sets is that they can be prepared to accord with the expected function of the measurement system and the nature of the measurements taken, in terms of data densities, signal-to-noise ratios, etc.

10.14 Back-to-back testing

In this form of testing two comparable software systems are tested with the same input. The output from each test is then compared — identical results are not usually expected when numerical testing is undertaken.

If the comparison can be automated, then it may be possible to run a large number of tests thus giving a high assurance that the two items produce similar results. Of course, one of the items under test is likely to be a version of known characteristics, while to other is the item being assessed.

This form of testing can be applied at a higher level than just a single software component. Indeed, the standard method of calibrating measurement system can be seen as a back-to-back test of a trusted measurement system against one to be calibrated.

This form of testing can also be used to test the soundness of numerical software within a measurement system. The results of the numerical calculation

from the measurement system are compared against the results of a piece of reference software which is a proven implementation of the numerical calculation. This is one of the methods used in the SS*f*M numerical software testing project, see [R-25]. Where there is no existing implementation, it may be possible to produce a reference implementation based on sound numerical analysis and using reliable software libraries. This reference implementation will presumably be less useful than the implementation used by the measurement system (e.g. it may be very slow) but it should give trusted output.

Example

In the SMART reliability study [SMART], this form of testing was used by testing a MatLab implementation against the C code within the measurement system. The test cases used were those derived from boundary value/equivalence partition testing.

Scope

The key requirement is a trusted system (or oracle) against which the testing can be applied. It is also necessary to have a comprehensive set of test cases from which a comparison can be made.

Cost/Benefits

Given a trusted implementation and a set of test cases, the method is cheap to apply. Conversely, if a trusted implemented is not available this method is probably not applicable.

In the case of testing numerical software, if there is not an existing reference implementation it may be possible (but expensive) to produce an implementation of the numerical calculations in isolation.

Conclusions

Since a trusted implementation is required and would be expensive to produce specially, the method depends upon the existence of such an implementation. On the other hand, given such an implementation, the method should probably be used.

Appendix A

Some Example Problems

A number of illustrative examples are collected here of problems that have been reported to NPL over a number of years. The exact sources are deliberately not given, even when they are known.

A.1 Software is non-linear

A simple measuring device was being enhanced to have a digital display. This was controlled by an embedded microprocessor, with the code produced in assembler. The product was then to be subjected to an independent test. The testers discovered, almost by accident, that when the device should have displayed 10.00 exactly, the actual display was nonsense. The fault was traced to the use of the wrong relational operator in the machine code.

The example contrasts with pre-digital methods of recording measurements in which the record is necessarily linear (or very nearly linear).

The example illustrates that the testing of software should include boundary conditions. However, only the most demanding standards actually *require* that such conditions are tested. For a four-digit display in this example, it should be possible to cycle through all the possible outputs to detect the error.

A.2 Numerical instability

The repeatability standard deviation of a weighing balance was required as part of a reference material uncertainty estimate. Successive weighings of a nominal 50 gramme weight produced a set of fifteen replicate values as follows: 49.9999 (1 occurrence), 50.0000 (5 occurrences), 50.0001 (8 occurrences) and 50.0002 (1 occurrence).

The processing of the data used an in-built "standard deviation" function operating to single precision (eight significant figures). Because the data could be represented using six significant figures, the user anticipated no difficulties. The value returned by the function, however, was identically zero.

The reason is that the function implements a "one-pass" algorithm that, although fast to execute, is numerically unstable. The standard deviation computation in this algorithm is based on a formula involving the difference between quantities which are very close for the above data values, thus causing

the loss of many figures. An alternative "two-pass" algorithm that first centres the data about the arithmetic mean and then calculates the standard deviation returns an answer for the above data that is correct to all figures expected. Unfortunately, the "one-pass" algorithm is widespread in its use in pocket calculators and spreadsheet software packages.

The example described above is concerned with the stability of the algorithm chosen for the required data processing. Numerical difficulties may also arise from the improper application of good algorithms. In one example, the processing software was to be ported from one (mainframe) platform to another (PC) platform. Although the platforms operated to similar precisions, and the same numerically stable algorithm was used (albeit coded in different languages), the results of the processing agreed to only a small number of significant figures. The reason was that the linear systems being solved were badly scaled and, therefore, inherently ill-conditioned, i.e., the solution unnecessarily depended in a very sensitive way on the problem data.

The lessons of this are: ensure the required data processing is stated as a well-posed problem; then use a stable algorithm to solve the problem.

A.3 Structural decay

A contractor is used to develop some software. They have very high coding standards which include writing detailed flow diagrams for the software before the coding is undertaken. The contractor corrects these diagrams to reflect the actual code before delivery to the customer. It is satisfactory to use flow charts to generate a program. But once the program is written, these charts become history (or fiction), and only charts generated from the program source are trustworthy.

The customer has tight deadlines on performing modifications to the software over the subsequent five years. For the first two amendments, the flow diagrams were carefully updated to reflect the changes to the code, but after that, no changes were made so that the flow diagrams were effectively useless. As a result, the overall 'design' provided by the contractor was effectively lost. The problem was that the 'design' was not captured in a form that could be easily maintained.

The conclusion from this is that for programs which have a long life, one must be careful to capture the design in a format that can be maintained. Hence it is much better to use design methods which support easy maintenance — hand-written flow charts are exactly what is *not* needed!

A more serious example of the same aspect is the use of programming languages which do not support high-level abstraction, for instance C as opposed to C++.

A.4 Buyer beware!

Professor W Kahn is a well-known numerical analyst who has also tested many calculators over many years. Several years ago, he illustrated an error in one calculator in the following manner: assume the calculator is used to compute the route to be taken by an aircraft in flying between two American cities, then

the error in the computation would result in the aircraft flying into a *specific* mountain.

Modern calculators are cheap and usually reliable. However, errors do occur. Hence the use of such machines in life-critical applications needs serious consideration. If the same calculations were being performed by software within the aircraft, then the very demanding avionics standard would apply [DO-178B]. Hence, when used for a life-critical application the same level of assurance should be provided by the calculator (however, it probably would not be cheap).

In the context of a measurement system, the above dilemma is critical. Many systems are sold without any specific application in mind. If the supplier makes no claim about such a system, it is appropriate to use the system in a safety application? If the measurements have a direct impact upon the safety of a system, then it is clear that the user has a responsibility to ensure that the system will not make the system unsafe (within the constraints of As Low As Reasonably Practical — ALARP).

With most physical devices, if high quality/assurance is required, there is likely to be a significant cost implication. Hence the supplier could be expected to charge a premium for an system qualified for use in safety application. However, with software, the situation is not so clear-cut, since the replication cost of software is effectively zero. Why should a mass-produced calculator be any less reliable/accurate than a similar device produced for a specialist market?

A.5 Long term support

Many applications at NPL are required to be available for over 10 years. For this length of time, problems can easily be encountered in ensuring the (hardware and) software will be properly supported.

Even with short-term support, problems can arise. Often, software is purchased some time before very extensive use is made of it. When such extensive use is applied, the initial support may have lapsed. Very many years ago, some defects were found in a small computer which resulted in the following response:

> *The software engineer who wrote the ZX81 BASIC is no longer with us, and his successors do not feel qualified to comment on your findings.*

Of course, this machine is no longer of interest, but the principle still applies. Companies can respond more easily and more quickly if faults are reported in the first few months of release, while the original design team is still available.

Users should estimate the expected life-time of an application and plan for the required support.

A.6 Measurement System Usability

In the quest for higher accuracy, it is sometimes forgotten that an measurement system will not always be used by the expert who designed it. With modern interactive software, it is possible to make the most complex of measurement systems 'easy' to use. However, reliability can be poor unless the design ensures that operator error is minimised.

Software within a measurement system can aid the user, but care is needed if the user is not to be misled. For instance, one modern spectral analyser provided an automatic means of separating adjacent peaks on a spectrograph, rather than being operator-driven. This worked satisfactorily most of the time, but the cases of failure were hard to detect and easy to miss. If automatic analysis is to replace manual analysis, one needs to ensure a comparable level of accuracy.

A.7 Tristimulus colour measurement

In this section, we consider an actual application with its own lessons to be learnt.

Three standard curves were developed by user trials in NPL in 1926 as part of work with the CIE. These curves X, Y and Z correspond to nominal spectra for Red, Green and Blue respectively.

When a tristimulus colorimeter performs a measurement the system performs a match of the spectra generated to apportion a best fit to the standard curves. The software then uses a set of standard equations to display the colour using one of very many scales. The scale used is the LCH where L is the Lightness, C the chroma and H the Hue.

A source code review of the internal code of this particular instrument was undertaken and a number of protocols were performed to test the functionality of the instrument.

After about 3 months routine use, some trials were undertaken to introduce a new product. In theory the maximum value for the L value is 100. The instrument obtained results of over 10000. Close examination of the sample showed it was generating a small stream of CO_2 bubbles through incomplete degassing of the solution. The minute stream of CO_2 bubbles was blocking the light path. The result was the detector was only picking up noise and was trying to match this to the standard curves.

The manufacturer added a software trap to detect transmission values of less than 0.5%.

Two months later on a routine test, several samples produced a lightness value of 100 and a chroma value of 10. This was investigated as the chroma should have been approximately 25. This time investigation showed the fault lay with the operator who had calibrated the instrument using a cell filled with the solvent as the 'white' standard. In this case the cell had not been properly cleaned before the calibration. It then was revealed that the manufacturers software contained a trap that if the sample's measured values were greater than obtained for the white standard they would be reduced to that of the white standard. The manufacturers theory was that as a white standard is defined as 100% of light is transmitted then you can never exceed 100% and any values exceeding 100% could only be noise.

The manufacturer removed this trap.

A.8 Balances

Another example application.

A new balance was launched and was assessing its suitability for use. One of the options when the balance was linked to a PC was to send a command to lock out parts of the keyboard. When we carried out some trials it was found that following a certain combination of commands a fault occurred and the balance keyboard was completely disabled.

The manufacturer issued a new software version.

Internally the user produced a check weighing package. Extensive test protocols were performed using a wide variety of balances from a number of suppliers. The software was issued to one of the user's customers using a balance option that had not been specifically tested. It was found that when a sample was added to the balance the weight displayed on the balance continued to display zero. Investigation showed that the user software was requesting a weight from the balance and used an algorithm to test that weight had increased. If the weight had not increased the software would request a further weight. The balance contained two chips of interest to the story. The first contained the software talking to the weighing cell. The second contained the general software that collected the weight from the first chip and sent it to the balance display and the PC. What was happening was that the software on the second chip was prioritising the data link to the PC. The manufacturers testing had been performed using a 486 PC. The user's customer was using a Pentium II processor and the additional speed at which data was requested was overwhelming chip 2. Chip 2 no longer had the capacity to talk to chip 1 to see if the weight had increased.

The manufacturer modified the software to solve the prioritisation.

Appendix B

Some Validation Examples

In this chapter, a number of examples are given of the approaches used in real measurement systems of demonstrating high quality for the software component. Each example references to main text and has been developed by using a draft of this Guide.

B.1 Measurement of flatness/length by a gauge block interferometer

NPL is the supplier of both the hardware and the software of this equipment.

The physics behind the measurement system is categorised as **validated physics**, see page 12.

The model of a measurement system (on page 13), applies to this measurement system.

The criticality of usage is **critical**, see page 18. If any fault was found, NPL would be very embarrassed. On the other hand, no life-critical application is known, nor seems likely.

The measurement system does not have to satisfy any specific legal requirement (see page 18).

The complexity of the control is rated as **very simple** or **simple**, see page 19.

The complexity of the data processing is rated as **modest** or **complex**, mainly because of the size (15-20,000 lines of C++), see page 19.

The answers to the key questions (see page 23) were:

1. It is known that end-to-end testing cannot provide sufficient confidence in the software, since it would be hard to locate any numerical instabilities in the processing by that means.

2. Raw data can be extracted and used for testing and is probably adequate for the length measurement, but not for flatness. However, many artifacts do not provide demanding cases for the software.

3. The software has been tested against an older system by back-to-back testing.

4. All errors and comments are logged, but since the equipment is only in trial use at the moment, error statistics are not very meaningful.
5. Any error in the control software cannot result in a false measurement.
6. The operator interface is not thought to be complex nor likely to put any measurement at risk.

Issues arising from the risk assessment questions (see page 33) were as follows:

- There is no record of a measurement system malfunction due to software (as opposed to errors in the software detected during development).
- The NPL software development process is internally audited and subject to an external audit by LRQA for ISO 9001.
- The measurement system does not require regulatory approval.
- The measurement system makes obvious internal checks and provides modest protection against operator error.
- The length computations are linear. The flatness calculations are non-linear, but have been checked by the NPL mathematics group for stability. The algorithm used for flatness is given in [Ref-Cheb, Cheb-Geo].
- An internal numerical error (overflow, etc) should be detected by the software.

As a result of the above, the software was rated as Software Integrity Level 3.

Issues arising from the general software questions (see page 37) were as follows:

- Since the measurement system has not yet been released to end customers, the provision of information (by NPL) has not yet arisen.
- No software reliability data is available, as the log of errors is an empty file at this point.
- The question of operator training and assessment of the user manual has not yet been addressed.
- The main aspect of the software which gives rise to concern is that of its size.

Issues arising from the Level 1 questions (see page 38) were as follows:

- No issues as NPL operates an ISO 9001 process.

Issues arising from the Level 2 questions (see page 39) were as follows:

- Software inspection is not undertaken, but code review is being done. The two people on the project review each other's code.
- There is a mathematical specification of the processing algorithms, and the code for these algorithms have been derived from the specification.

- During the development, the main processing code was tested (informally) to branch level.

Issues arising from the Level 3 questions (see page 39) were as follows:

- Regression testing has been applied from the start, and the results have been recorded.
- The variant of stress-testing for image processing has been used. This testing revealed a weakness in the specification.
- All known (and documented) errors are trivial.
- The system is too young to suffer yet from structural decay.

Issues arising from the Level 4 questions (see page 40) were as follows:

- As noted above, informal branch testing was undertaken on the main algorithms. This should imply 100% statement coverage.
- No Formal Specification has been produced.
- Static analysis has not been undertaken.
- It is not thought that there is a role for accredited testing of the software.
- An attempt was made to detect memory leaks in the software.

The Software Engineering techniques listed in this Guide were applied as follows:

Software inspection: (see page 41.) No, but code review undertaken.

Component testing: (see page 43.) Yes, on processing algorithms.

Regression testing: (see page 45.) Yes.

Validation suite: (see page 46.) Not applicable.

System-level testing: (see page 47.) Yes.

Numerical stability: (see page 48.) Yes.

Mathematical specification: (see page 50.) Yes.

Formal Specification: (see page 51.) No.

Independent audit: (see page 52.) (NPL audit, and external ISO 9001 audit.)

Stress testing: (see page 52.) Yes, for image processing part.

Static analysis: (see page 53.) No.

Reference test sets: (see page 54.) Yes, for flatness algorithms.

Back-to-back testing: (see page 56.) Yes, with older software.

Conclusions

The validation of this software is generally consistent with the Software Integrity Level of 3. One of the difficulties facing the development is that only two people are involved, making a comprehensive independent review effectively impossible. Code review of each other's code is being undertaken as a result of an internal NPL audit. The size of the code makes this a large task.

B.2 Electrical measurement resistance bridge

The device with its software provides the primary service for NPL to calibrate resistance measuring devices. The measurement system is entirely for in-house use by NPL, and hence NPL is the supplier and the sole user.

Oxford Instruments are licenced by NPL to manufacture a similar device, but this device has different software and is not considered here.

The physics behind the measurement system is categorised as **validated physics**, see page 12.

The model of a measurement system (on page 13), applies to this measurement system.

The criticality of usage is **critical**, see page 18. If any fault was found, NPL would be very embarrassed, especially with the international intercomparisons.

The measurement system does not have to satisfy any specific legal requirement (see page 18).

The complexity of the control is rated as **complex**, see page 19. The control consists of 10,000 lines of Turbo-Pascal which uses its own macro language to aid the control.

The complexity of the data processing is rated as **simple**, see page 19. Only two stored calibration constants are used in a linear calculation. The data processing part is a separate program from the control logic, communicating via a file.

The answers to the key questions (see page 23) were:

1. Very good confidence in the measurement system can be established by end-to-end testing. Standard artifacts can be used for this purpose and the number of individual tests that need to be undertaken is widely recognised.

2. Since the control and processing programs are separate, they can be validated independently.

3. The software has been in use for 10 years in various versions.

4. No errors has been recorded which gave an incorrect measurement. The main problems have been in the operator interface to the control program.

5. It seems hard for the control software to produce an undetected error in a measurement. Only a few lines copy the basic data into a file for processing by the data processing program.

6. Only qualified staff at NPL can use the measurement system for calibration work.

Issues arising from the risk assessment questions (see page 33) were as follows:

- As noted above, the control software has essentially no impact on the measurements themselves.
- The data processing software in classified as **simple** and hence straightforward to validate.
- Since the measurement system is only used by NPL staff, the control software does not need to handle many cases of mis-use. (The Oxford Instruments version is quite different in providing simpler control with fewer user options.)

As a result of the above, the software was rated as Software Integrity Level 1. Note that this is a lot lower that the previous example (3).

Issues arising from the general software questions (see page 37) were as follows:

- With the supplier being the sole user, the provision of information is not a problem.
- The raw data is in a file used by the data processing program.
- An independent check has been made on the data processing program by comparing it with the similar program developed for the Oxford Instruments device.
- Completeness of the system testing has not been measurement by, for instance, test coverage. This is not thought to be a problem due to the unlikelihood of the control software giving a problem.

Issues arising from the Level 1 questions (see page 38) were as follows:

- Since the basic design of the software is ten years old, much of the design is actually embedded in the code itself, rather than in a separate document. However, the complex design aspects are in the control software which does not impact the basic measurements.
- The only recent change to the computing environment has been the compiler, which caused the system to be re-tested.
- The system is maintained by a physicist rather than a programmer. This is adequate in this instance, since the system is assessed at a Software Integrity Level of 1. If a higher level of integrity was required, additional skills in software engineering would be needed.

Issues arising from the Level 2 questions (see page 39) were as follows:

- It was noted that a code review should have been undertaken on the software.
- The system testing applied to the control software did not provide any measurement of testedness, such as statement coverage.

Issues arising from the Level 3 questions (see page 39) were as follows:

- Only internal audits have been undertaken.
- It is thought that structural decay is a minor problem due to the age of the software.

Issues arising from the Level 4 questions (see page 40) were as follows:

- Specific methods recommended at this level have not been applied.

The Software Engineering techniques listed in this Guide were applied as follows:

Software inspection: (see page 41.) No.

Component testing: (see page 43.) No.

Regression testing: (see page 45.) Yes, for data processing program.

Validation suite: (see page 46.) Not applicable.

System-level testing: (see page 47.) Yes, and gives required confidence in the measurement system.

Numerical stability: (see page 48.) Yes.

Mathematical specification: (see page 50.) Yes.

Formal Specification: (see page 51.) No.

Independent audit: (see page 52.) (NPL audit, and external ISO 9001 audit.)

Stress testing: (see page 52.) No.

Static analysis: (see page 53.) No.

Reference test sets: (see page 54.) No.

Back-to-back testing: (see page 56.) Yes, NPL data processing software against the similar software for the Oxford Instruments device.

Conclusions

The risk analysis shows that this software is of low risk and hence a Software Integrity Level of 1 is appropriate, even though a measurement error would be an embarrassment to NPL. The key aspect is that testing the measurement system as a black-box gives a high assurance of the system, including the software relevant to the measurement.

The validation of the software undertaken by NPL is generally consistent with the Software Integrity Level of 1.

B.3 Mean renal transit time

This medical application consists of a gamma camera and associated software for measurement of mean renal transit time. The diagnostic imaging procedure in question is called a *renogram*, and the medical specialty of which it is a part is called *nuclear medicine*.

The patient is injected (in an arm vein) with an appropriate radiopharmaceutical, whose passage through the body can be observed by the use of a large field of view (approx 50×40cm) radiation detector called a gamma camera. In this application, the detector is positioned under the patient (lying supine on a low attenuation imaging couch), covering the posterior aspect of the lower chest and abdomen, centred on the kidneys. Immediately after injection, a sequence of digital images (10 second frames) is recorded for a period of 20 minutes, showing the passage of the injected material into, and out of, the organs within the field of view (heart, spleen, liver, kidneys and bladder). In this case, the organs under scrutiny are the kidneys.

The first stage of the image processing involves the creation of so called regions-of-interest (ROI) around particular organs and structures (both kidneys, heart, and two 'background' areas). This is usually done manually, using a mouse/cursor. In image processing parlance this would be called 'segmentation'. Each ROI is then applied, in turn, to each of the 120 frames, and the 'information density' (in this case count rate) within each ROI plotted as a function of time. Five curves are thus produced. These curves are then analysed further to calculate the mean renal transit time for each kidney. The use of this model/algorithm/software simulates the effect of direct injection into the renal artery, which is extremely invasive and requires an anaesthetic. Intravenous injection, by contrast, is minimally invasive and is happily tolerated by the vast majority of patients. The passage of a non-reabsorbable solute (of which the radiopharmaceutical injected is an example) through the kidney is slowed in various renal disorders, and the mean renal transit time (of that solute) is thus prolonged. The measurement of mean renal transit time can be used to diagnose kidney disorders such as upper urinary tract obstruction, which is usually caused by narrowing of the junction between kidney and ureter.

The information below has been provided by Mr P.S.Cosgriff who is a user of the system, but also has a detailed knowledge of its workings. He has written some software that can perform part of the analysis, although most of the system being considered is supplied by other (commercial) parties. The gamma camera was supplied by one company, and the image processing software by another.

The physics behind the instrument is categorised as **Physics can be modelled**, see page 12.

The model of an instrument (on page 13), applies to this instrument, but note the degree of interaction as noted above.

The criticality of usage is **Potentially life-critical**, see page 18. The patients doctor (hospital consultant) makes the final decision regarding the use of the results in deciding treatment.

The instrument has to satisfy the specific legal requirements (see page 18) in Medical Device Regulations 1996 (as a Class 2A product).

The complexity of the control is rated as **Complex**, see page 19.

The complexity of the data processing is rated as **Complex**, see page 19.

The answers to the key questions (see page 23) were:

1. Due to the complexity of both the control and data processing software, only limited assurance can be obtained from end-to-end testing. It is also difficult to produce appropriate models (hardware 'phantoms') for testing due to the difficulty in simulating a complex biological system.

2. The data from the camera is essentially separate and is typically available in a format that can be processed by any nuclear medical image processing. In consequence, the gamma camera and the software can be considered separately.

   Naturally, the camera can be considered as an instrument it its own right, but this is not considered further here.

3. Yes, similar instruments are available for which some reliability data is available.

4. An error log is available from the bug-fix information sent out by the software supplier.

5. False measurements are certainly possible. For instance, if the patient moves and the operator fails to take this into account, the image processing software could fail to locate the kidney correctly.

6. The operator interface is complex and only suitably trained staff can operate the equipment.

Issues arising from the risk assessment questions (see page 33) were as follows:

- At worst, an instrument malfunction could cause inappropriate treatment to the patient.

- The gamma camera performs some built-in tests, but the software does not.

- The control functions cannot be effectively protected against operator error. (Hence the need for skilled operators.)

- The computation is complex and non-linear. The final calculation after the image processing uses differences and requires that the initial data points are heavily smoothed to avoid numerical instabilities.

- It is unclear how robust the software is with regard to issues like numerical overflow.

For this example, the Software Integrity is taken as Level 4. SIL3 may be more realistic. A drug infusion pump connected to a patient on an Intensive Care Unit would certainly be SIL4, but the software contained would be probably 100 times less complex than taht described above and would also probably have hardware interlocks to prevent catastrophic failure. On balance, therefore, SIL 4 is thus considered reasonable for the measurement of mean renal transit time.

Issues arising from the general software questions (see page 37) were as follows:

- An independently specified file format is available which allows a user or manufacturer to test the software without the use of the gamma camera.

The questions concerning software development cannot usually be answered, since the information available is from a user.

Issues arising from the Software Integrity Level 1 questions (see page 38) were as follows:

- There is an error log and evidence of clearance of errors.
- It is unclear how the exact version of the software is identified (which is non-trivial due to updates being applied by users). The Notifying Body oversight should check for this.

Issues arising from the Level 2 questions (see page 39) were as follows:

- No information available.

Issues arising from the Level 3 questions (see page 39) were as follows:

- No information available.

Issues arising from the Level 4 questions (see page 40) were as follows:

- No information available.

The Software Engineering techniques listed in this Guide were applied as follows (little information is available at this point):

Software inspection: (see page 41.) —

Component testing: (see page 43.) —

Regression testing: (see page 45.) —.

Validation suite: (see page 46.) Accredited testing to be undertaken under [COST B2].

System-level testing: (see page 47.) —.

Numerical stability: (see page 48.) —.

Mathematical specification: (see page 50.) —.

Formal Specification: (see page 51.) —.

Independent audit: (see page 52.) As part of type approval.

Stress testing: (see page 52.) —

Static analysis: (see page 53.) —

Reference test sets: (see page 54.) —

Back-to-back testing: (see page 56.) —

Conclusions

The risk analysis shows that there is a moderate risk of inappropriate treatment should the complex software produce an incorrect result. The ability to test the software by means of 'software phantoms' has led to a proposal to develop a formal certification process for this type of medical software [COST B2].

Bibliography

[AOAC] S.L.R. Ellison, M.G. Cox, A.B. Forbes, B.P. Butler, S.A. Hannaby, P.M. Harris and Susan M. Hodson. Development of data sets for the validation of analytical instrumentation. J. AOAC International, 77:777-781, 1994. 10.13

[BS 7925] British Computer Society Specialist Group in Software Testing. Standard for Software Component Testing (Working Draft 3.0). Glossary of terms used in software testing (Working Draft 6.0). October 1995. Now revised as a BSI standard, BS7925 part 1 and 2 available from BSI. 10.3

[Cheb-Geo] R Drieschner. Chebyshev approximation to data by geometric elements. Numerical algorithms. Edited by C Brezinski. Volume 5. pp509-522. J C Baltzer. 1993. B.1

[Chem] R. G. Brereton. Chemometrics: Applications of Mathematics and Statistics to Laboratory Systems. Ellis Horwood, Chichester, England, 1990. 10.13

[COST B2] COST B2: the quality assurance of nuclear medicine software. K E Britton and E Vauramo. European Journal of Nuclear Medicine. 1993. vol 20. pp815-816. Details available at `http://www.phb-mpd.demon.co.uk/index.html` 10.5, B.3, B.3

[Design] M.G. Cox and P.M. Harris. Design and use of reference data sets for testing scientific software. Analytica Chimica Acta 380, pp 339 - 351, 1999. 10.13

[DO-178B] Software Considerations in Airborne Systems and Equipment Certification. Issued in the USA by the Requirements and Technical Concepts for Aviation (document RTCA SC167/DO-178B) and in Europe by the European Organization for Civil Aviation Electronics (EUROCAE document ED-12B). December 1992. 2, 9, A.4

[Dyson] N. Dyson. Chromatographic Integration Methods. Royal Society of Chemistry, 1990. 10.13

[EN45001] EN 45001. General criteria for the operation of testing laboratories. CEN. June 1989. 2

[FDA] ODE Guidance for the Content of Premarket Submission for Medical Devices Containing Software. Draft, 3rd September 1996. 2

[FM] B A Wichmann. A personal view of Formal Methods. National Physical Laboratory. March 2000. Available free on the Internet at `http://www.npl.co.uk/ssfm/download/index.html` 10.9

[GAMP] Supplier Guide for Validation of Automated Systems in Pharmaceutical Manufacture. ISPE 3816 W. Linebaugh Avenue, Suite 412, Tampa, Florida 33624, USA. 5.5

[Golub] G.H. Golub and C.F. Van Loan. Matrix Computations. John Hopkins University Press, Baltimore, MD, USA, 1996. Third edition. 10.13

[Grade-sets] M.G. Cox. Graded reference data sets and performance profiles for testing software used in metrology. In P. Ciarlini, M.G. Cox, F. Pavese, and D. Richter, editors, Advanced Mathematical Tools in Metrology III, pages 43-55, Singapore, 1997. World Scientific. 10.13

[HF-GUIDE] Safety-related systems — Guidance for engineers. Hazards Forum. March 1995. ISBN 0 9525103 0 8. 1

[HSC] The use of computers in safety-critical applications. Final report of the study group on the safety of operational computer systems. HSC. London. ISBN 0-7176-1620-7. 1998. 2

[IEC 601] IEC 601-1-4: 1996. Medical electrical equipment — Part 1: General requirements for safety 4: Collateral Standard: Programmable electrical medical systems. 2

[IEC 61508] IEC 61508: Parts 1-7, Draft. Functional safety: safety-related systems. Draft for public comment, 1998. (Part 3 is concerned with software which is the relevant part for this Guide.) 2, 5.3, 7, 9

[IEE-PLC] SEMSPLC Guidelines: Safety-Related Application Software for Programmable Logic Controllers. IEE Technical Guidelines 8: 1996. ISBN 0 85296 887 6. 7

[INSPECT] T Gilb and D Graham. Software Inspection. Addison-Wesley 1993. ISBN 0-201-63181-4. 10.1

[ISO 25] ISO/IEC Guide 25: 1990. General requirements for the competence of calibration and testing laboratories. (This is to be replaced in 1999 by ISO/IEC 17025 with the same title.) 2, 10.10

[ISO 9000-3] ISO/IEC 9000-3: 1991. Quality management and quality assurance standards — Part 3: Guidelines for the application of ISO 9001 to the development, supply and maintenance of software. 2, 5.5

[ISO 9001] EN ISO 9001:1994, Quality systems — Model for quality assurance in design, production, installation and servicing. 2, 4, 5.5, 10.10, 10.10

[ISO 9126] ISO/IEC 9126:1991, Information technology — Software product evaluation — Quality characteristics and guidelines for their use.

[ISO 10360-6] ISO/DIS 10360-6. Geometrical product specifications (GPS) — acceptance test and reverification test for coordinate measuring machines (CMM). Part 6: Estimation of errors in computing Gaussian associated features, 1999. International Standards Organisation proposed draft standard. 2, 5.4, 10.7, 10.13

[ISO 12119] ISO/IEC 12119: 1993, Information technology — Software packages — Quality requirements and testing. 10.6, 10.6

[ISO 12207] ISO/IEC 12207: 1995. Information technology — Software life cycle processes. 5.5

[ISO 13233] ISO/IEC TR 13233: 1995. Information Technology — Interpretation of Accreditation Requirements in ISO/IEC Guide 25 — Accreditation of Information Technology and Telecommunications Testing Laboratories for Software and Protocol Testing Services. 2, 5.4

[ISO 17025] ISO/IEC 17025: 1999. General requirements for the competence of testing and calibration laboratories. 2

[LIMS] R Fink, S Oppert, P Collison, G Cooke, S Dhanjal, H Lesan and R Shaw. Data Measurement in Clinical Laboratory Information Systems. Directions in Safety-Critical Systems. Edited by F Redhill and T Anderson, Springer-Verlag. pp 84-95. ISBN 3-540-19817-2 1993. 3.4

[LINPACK] LINPACK User's Guide. J J Dongarra, C B Moler, J R Bunch, and G W Stewart. SIAM, Philadelphia. 1979. 10.7

[Lit 92] B Littlewood and L Strigini. The Risks of Software. Scientific American. November 1992. 9

[M10] NAMAS Accreditation Standard. M10. NAMAS. 1992. 2, 10.10

[MISRA] Development Guidelines For Vehicle Based Software. The Motor Industry Software Reliability Association. MIRA. November 1994. ISBN 0 9524156 0 7. 2

[Monitor] A Coombes, L Barroca, J S Fitzgerlad, J A McDermid, L Spencer and A Saed. Formal Specification of an Aerospace system: the Attitude Monitor. Chapter 13, Application of Formal Methods. M G Hinchey and J P Bowen. Prentice-Hall 1995. 10.9

[NAG] The NAG Fortran Library. The Numerical Algorithms Group Ltd., Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, UK. 10.7

[NIS37] A Guide to Managing the Configuration of Computer Systems (Hardware, Software and Firmware) used in NAMAS Accredited Laboratories. NIS37. NAMAS, October 1993. 2, 4

[NIS40] Storage, Transportation and Maintenance of Magnetic Media. NIS40. NAMAS, October 1990.

[NIS41] Use of Word Processing Systems in NAMAS Accredited Laboratories. NIS41. NAMAS, November 1991. 3.4

[NORDTEST] Guidelines for assessment of software in microcomputer controlled equipment for safety-related systems. NT TECHN Report 287. May 1995. 2

[NPL-LIB] The National Physical Laboratory's Data Approximation Subroutine Library. G T Anthony and M G Cox. In J C Mason and M G Cox, editors, Algorithms for Approximation, pages 669-687. Clarendon Press, Oxford, UK. 1987. 10.7

[Opt] P.E. Gill, W. Murray and M.H. Wright. Practical Optimization. Academic Press, 1981. 10.13

[Opt-test] B.P. Butler, M.G. Cox, A.B. Forbes, S.A. Hannaby, and P.M. Harris. A methodology for testing classes of approximation and optimisation software. In R. F. Boisvert, editor, The Quality of Numerical Software: Assessment and Enhancement, pages 138-151, London, 1997. Chapman and Hall. 10.13

[R-25] H.R. Cook, M.G. Cox, M.P. Dainton and P.M. Harris. A methodology for testing spreadsheets and other packages used in metrology. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme. NPL Report CISE 25/99, September 1999. 2, 10.13, 10.14

[R-26] H.R. Cook, M.G. Cox, M.P. Dainton and P.M. Harris. Testing spreadsheets and other packages used in metrology: A case study. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme. NPL Report CISE 26/99, September 1999. 10.13

[R-27] H.R. Cook, M.G. Cox, M.P. Dainton and P.M. Harris. Testing spreadsheets and other packages used in metrology: Testing the intrinsic functions of Excel. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme. NPL Report CISE 27/99, September 1999. 10.13

[Ref-Cheb] G T Anthony, H M Anthony, B P Butler, M G Cox, R Drieschner, R Elligsen, A B Forbes, H Gross, S A Hannaby, P M Harris and J Kok. Reference software for finding Chebyshev best-fit geometric elements. Precision Engineering. Vol 19, pp28-36. 1996. B.1

[Ref-sets] B.P. Butler, M.G. Cox, S.L.R. Ellison, and W.A Hardcastle, editors, Statistics Software Qualification: Reference Data Sets. Royal Society of Chemistry, Cambridge, ISBN 0-85404-422-1. 1996. 10.13, 10.13

[RISKS] Guidelines on Risk Issues. The Engineering Council. February 1993. ISBN 0-9516611-7-5. 4

[SA-Survey] B A Wichmann, A A Canning, D L Clutterbuck, L A Winsborrow, N J Ward and D W R Marsh. An Industrial Perspective on Static Analysis. Software Engineering Journal. March 1995, pp69-75. 10.12

[SERB] J A McDermid (Editor). Software Engineer's Reference Book. Butterworth-Heinemann. Oxford. ISBN 0 750 961040 9. 1991. 10

[SMART] A Dobbing, N Clark, D Godfrey, P M Harris, G Parkin, M J Stevens and B A Wichmann, Reliability of Smart Instrumentation. National Physical Laboratory and Druck Ltd. August 1998. Available free on the Internet at `http://www.npl.co.uk/ssfm/download/index.html` 2, 2, 3.6, 4.2, 10.3, 10.8, 10.12, 10.14

[SSfM] D Rayner. National Measurement System Software Support for Metrology Programme, 1998-2001. `http://www.npl.co.uk/ssfm/`

[SSfM-Model] National Measurement System Software Support of Metrology Programme, Theme 1: Modelling Techniques. `http://www.npl.co.uk/ssfm/model/index.html` 3.1, 3.3

[STRESS] B A Wichmann. Some Remarks about Random Testing. National Physical Laboratory. May 1998. Available free on the Internet at `http://www.npl.co.uk/ssfm/download/index.html` 10.11, 10.11, 10.11

[UKAS-61508] Study to investigate the feasibility of developing an **accreditable** product-based scheme of conformity assessment and certification based on satisfying the requirements of International Standard IEC 1508. UKAS. 1997. Available free on the Internet at `http://www.npl.co.uk/npl/cise/UKAS/index.html` 2

[UKAS-TickIT] Use of TickIT as a benchmark for software. UKAS. CIS1 November 1999. Available free on the Internet at `http://www.ukas.com/pdfs/TickIT.pdf` 5.5

[VDM-SL] J Dawes. "The VDM-SL Reference Guide". Pitman Publishing. 1991. ISBN 0-273-03151-1 10.9

[WELMEC-2.3] Guide for Examining Software (Non-automatic Weighing Instruments). WELMEC 2.3, January 1995. Available free on the Internet at `http://www.welmec.org/publicat.htm` 2, 4.1

[WELMEC-7.1] Software Requirements on the Basis of the Measuring Instruments Directive. WELMEC 7.1, January 2000. Available free on the Internet at `http://www.welmec.org/publicat.htm` 2

[Y2000] Test Report: Annodeus Ltd, Diagnostic and Fix programs. National Physical Laboratory. May 1st 1998. 10.6

[Z] Spivey J M. Understanding Z, CUP 1988. 10.9

Index