NPL Report CISE 16/98

March 1998

Objective Test Criteria: Some Proposals for Bespoke Software

Brian A Wichmann

E-mail: Brian.Wichmann@npl.co.uk
Centre for Information Systems Engineering
National Physical Laboratory
Teddington
Middlesex
United Kingdom
TW11 0LW

Abstract
The report details proposals made as a result of a study for the Ministry of Defence. The proposals are for objective test criteria to provide greater assurance for bespoke software. Consideration is given to the use of test metrics in this regard.

ISSN 1361–407X

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

Approved on behalf of Managing Director, NPL
by Dr D Rayner, Centre for Information Systems Engineering

Contents

1 Introduction

The paper describes the results of a short study being undertaken for the Ministry of Defence on **Objective Test Criteria** in which the main area of concern is bespoke software. The study was undertaken by the author during the period June 97 to December 97.

2 The Problem

The Ministry of Defence procures a large volume of bespoke software. The Ministry wishes to improve the assurance of the software it purchases to minimise difficulties found during acceptance and subsequently. Such difficulties result in a lack of confidence in the software and the supplier by the Ministry.

Of course, with large, complex systems, it is impractical to achieve 100% testing. Equally, the supplier will have subjected the deliverables to acceptance tests and therefore some functionality will have been demonstrated.

The question arises as to whether the software supplier would be prepared to sign up to some objective test criteria which could serve to demonstrate that the software had been tested in the specific manner and reduce the risk that defects are found by the Ministry during acceptance testing or soon after delivery.

Although this study is for the Ministry of Defence, it is my understanding that the Ministry would like its procedures to align with industry as far as is practical. In consequence, approaches to this problem that have only been applied in the civil sector are certainly relevant. In any case, many companies operate in both the military and civil domains and hence convergence between the two is advantageous.

3 Scope of the Study

The scope of this study are the large, bespoke systems, purchased by the Ministry. These can be very complex, have many novel aspects, and therefore will present a serious challenge to achieve the reliability and functionality that the Ministry requires.

Note that this study is *not* directly concerned with safety and security software. Safety-critical software requirements already have objective testing requirements specified [12, 10], and the security area is covered by assessment criteria [9]. However, the focus on high integrity software does indicate what can be achieved given appropriate resources. In consequence, the safety/security community has been given an opportunity to comment on the proposals made here. For those concerned with safety, it is probably best to think of this study being aimed at Levels 1 and 2 on the scale in IEC 1508 [5].

Some of the methods of assurance, such as the application of ISO 9001 [7], fall outside the scope of this study. Since registration to ISO 9001 is effectively a requirement of the Ministry of

Defence, this study assumes that suppliers have a Quality Management System which conforms to that standard.

Also excluded from this study is Commercial Off The Shelf (COTS) software, since the assumption is that the supplier is developing the software for the customer. However, this does not preclude some COTS components being used within the delivered product. Of course, if the COTS items were objectively tested, especially by a third party, then this could be used as evidence for its fitness-for-purpose. COTS is excluded since too many aspects would not necessarily have any visibility, such as white-box testing.

4 Objective Testing

Outside the area of software, objective testing is widely used as a quality assurance method, usually relying upon independent test laboratories [6, 4, 14].

For those unfamiliar with other branches of engineering in which objective testing is routinely undertaken, it is worth quoting an example. For major civil engineering projects, it is vital that the concrete used sets correctly and has the right mechanical strength. Hence the contractor is typically required to send samples of the concrete mix as small cubes to independent testing laboratories. The cubes are tested to destruction, but if they do not have adequate strength, the contractor will be required to rebuild the structure (at a potentially huge cost). In contrast, stress testing of software often reveals significant defects [17].

The software standards quoted here ensure objectivity which may be impractical in the context of complex software systems (due, for instance, to the difficulty of objective functional testing, see below). This may imply that the use of independent test laboratories is not appropriate, but the principle of objective testing could still be applied even when the testing is undertaken by the supplier.

Two examples are available of standards to specific objective test techniques which are relevant to this study as follows:

ISO package testing. This ISO standard defines black-box testing of a software package based largely upon functional testing [8]. The functional testing is based upon a detailed analysis of the product description – both the marketing material and the user documentation. White-box testing, such as statement coverage, is explicitly excluded.

BCS component testing. This British Computer Society standard [2] specifies objective criteria for testing software components. It includes the classical white-box and black-box testing methods.

These two standards cover only part of the concerns of this study since the ISO standard only handles software packages, and the BCS one only components. Nevertheless, these two standards serve to demonstrate that objective testing can be applied to software in a practical manner.

The author has undertaken a study recently on the validation of software in scientific instruments [16]. The Guide also gives examples of methods of increasing user confidence in software which is potentially useful for this study.

Half way through this study, the Ministry of Defence issued a standard having a direct impact upon this area [11]. Clearly, the purpose of objective testing for the Ministry is to provide evidence that would support a claim for reliability at a level that is perceived as appropriate for the application. This new Guide provides a framework for a rigorous approach to reliability whereas the issue that this study is attempting to address is the contribution objective testing can make to that framework.

The new Guide poses specific problems for this study (apart from being issued during the project):

- It cannot be assumed that the Guide will be applied to existing projects. Moreover, due to the longevity of MoD projects, it might be many years before this could reasonably be applied.
- Being a Guide rather than a full Standard, its application is not mandated. A working assumption would be that new project proposals made to the Ministry would apply those parts which the contractor thought appropriate.
- The Guide appears to be a counsel of perfection and therefore its practical application might follow the general philosophy but be modest in the specifics. (This is not a criticism of the Guide or its potential application since a list of specific requirements might encourage compliance with those specifics *at the cost of actual software quality*.)

5 The leading question for suppliers

The initial paper produced as part of this study raised the following question with suppliers:

> *As a supplier of software, are there any methods of objective testing which you could apply and would be prepared to exploit as a means of giving increased confidence to your customer?*

This question was first sent to the York University safety-critical systems e-mail list. This list contains many people very concerned with software quality issues, albeit in the context of safety for which testing requirements are often specified in standards. Subsequently, some companies were targeted for interview and further dialogue from which the proposals below have been derived.

6 Some Analysis

Attempting to improve customer confidence is very difficult. We lack a basic understanding of the software engineering process such as the most cost-effective methods to apply. The use of

computers to aid in the process is typically more effective in the coding phase rather than the specification phase. Yet the commonly accepted statistic is that 85% of the problems are with the specification and that their correction is more expensive than errors in coding.

There is a danger that studies like this encourage the Ministry to require technique X when the optimal strategy depends upon the project, the company expertise, and the other techniques in use. Software engineering requires skilled staff with good judgement — so any attempt at de-skilling or depending too heavily on standard recipes is unlikely to be effective. Moreover, unlike concrete mixing, the technology and products are changing fast. Fine-tuning based upon yesterday's experience will not be optimal for today, let alone tomorrow.

A number of issues arise from this study which must be considered before any realistic proposals can be considered:

Process or Product? There are two aspects to consider: the actual delivered item of software as a Product and also the Process by which it was constructed. The example of testing of concrete cubes given above would appear to suggest that only the Product need to be considered. Moreover, the black-box methods in the ISO package standard and the BCS component testing standard are clearly Product-based testing methods.

However, this study is concerned with bespoke software for which there is a defined process which could have some visibility to the customer (if that is seen as being advantageous to the supplier).

The Ministry of Defence has recently published its guidance to its own project managers which advises on how the ideal software development process should appear. The relevant section on testing [13] for this study reads:

> *Each individual requirement in the requirements definition must be amenable to an objective test that is feasible both technically and economically. This applies both to requirements that are stated directly and to those that are invoked by reference to other documents such as standards. Special care is needed with non-functional requirements.*

In addition, metrics on the software development process may be provided, giving additional insight into the suppliers' activities.

Objective testing against ITSEC [9] for security is accepted, but this is an independent evaluation process which incurs additional costs which is probably not acceptable in many other areas of defence procurement.

In consequence of the above, proposals which address the problem of objective testing by means of adding measures to the process would be feasible.

Risk analysis. In the same way that it is assumed that a Quality Management System is being employed by the supplier, we assume that a risk analysis has been undertaken for the project. Since safety-critical software is not effectively within the scope of this study, the risks arise from issues other than safety. In practice, the main concern would be business

success, since any major problem with the software could adversely affect further orders from the same customer. For many an equally important concern is the risk of being sued for damages caused to others (financial rather than safety).

It is hoped that acceptance of the recommendations given here will be seen as a method of reducing the risks. In any case, the risk analysis undertaken by the developer should be visible to the purchaser. However, since risk analysis is beyond the scope of this study, we do not provide guidance on how this risk analysis information should be developed and supplied to the purchaser.

For larger systems, the risk associated with some components will be different enough to justify different development strategies. This then is influenced by the next topic below.

Design for reliability. The Defence Standard 00-42, part 2 [11] is concerned with designing software to specific reliability targets. In consequence, at first sight it might appear that adherence to this standard might solve the basic problem, even if objective testing as such had no specific rolê.

Unfortunately, assurance and reliability are *not* the same. Since reliability can only be measured after the software has been written, it has a limited rolê in assurance. The main concern here is assurance, especially with early releases of software since no credible means are available for assessing the reliability of the delivered product.

There is also a fundamental problem with reliability of software — one does not design to specific reliability targets but rather places more or less effort to achieve zero defects in software.

Given that a large system is being produced, but with high reliability targets, then if system upgrades and specification changes are frequent, then it is hard to see how the reliability can be attained without introducing 'firewalls' or some scheme which provides partial fault tolerance.

Fault tolerance. If there is a requirement for fault-tolerance within the agreed specification of the product, then that can be considered like any other aspect of the specification.

However, it may well be that a supplier has some freedom as to how much fault tolerance to provide within the delivered system. This then provides a dilemma since the customer perception can vary. One could have two systems which are regarded as being of similar quality: one with very good reliability but no fault tolerance, the other which is less reliable but has a high degree of fault tolerance (so that the system recovers from most internal software faults).

The problem we have here is that the perception of quality is similar, but the reliability of the two solutions is very different.

Functional testing. This is a key issue. Clearly, if functional testing could be seen to be objective, repeatable and reproducible, then the whole problem would be solved. This is not the case, and hence an analysis must be undertaken to see what is feasible.

For components, equivalence partition testing is viable and forms one of the techniques in the BCS standard. But as a component is the smallest item of software with a separate specification (in that standard), the internal complexity is limited, which essentially makes the technique viable. Since we are dealing with complete systems, the BCS standard is not really applicable here.

'Complete' functional testing is undertaken in the safety context, sometimes called 'validation testing'. Even for a relatively simple system, the documentation behind this form of testing can be many hundreds of pages. It must include the specification of each test case, the expected results and a record of the actual test. Even with this level of documentation, the degree of 'completeness' is hard to establish which is why structural testing is also required (by safety standards, say).

Some insight can be gained into the issue by considering an item of software often used in critical contexts — e.g. an Ada compiler. A simple demonstration of the functionality of the compiler is provided by the 4,000 or so tests in the Ada validation suite. All serious compilers are validated and hence pass these tests.

In addition, compiler vendors have their own regression tests which over the years, can become substantially larger than the validation suite. In spite of using both the validation suite and their regressions tests, surprisingly simple errors nevertheless arise [15]. Stress testing compilers also reveals many errors [17] (although Ada compilers appear to be better than others).

A major problem is that we do not have adequate metrics for functional testing. More practically, we have ignored problems arising from ambiguities in the specification. An interesting example which seems to me indicative of the problem arose in a discussion with one supplier: the system in question had a number of terminals and in consequence, data could be associated with a terminal or be global. In one case, the data was implicitly assumed in the users' specification to be global, but should have been associated with the terminal. Conventional host testing did not reveal the problem since nobody thought to provide a parallel test with two terminals using the data. Once the problem is seen, it is easy to correct. Natural language does not handle concurrency well, so it is easy to get a specification error when concurrency is implicit rather than explicit.

Another aspect is that tool support for structural testing is much more mature than tools for functionality testing. This encourages developers to undertake structural testing when perhaps functional testing would be more cost-effective. As part of this study, I have been sent material on the tool SoftTest [20] which appears to address the real concerns with specifications — by means of an ambiguity analysis and cause-effect graphing. However, it was not possible to make an analysis of the contribution the tool could provide. Providing a better link between the specification of software and the result of objective testing would be highly advantageous.

Enquiries were made of the functional testing undertaken to ISO 12119 in Germany, but no information has been forthcoming at this time.

Hence we conclude that *functional testing* itself does not provide a complete solution to the problems faced here, but should be seen as an aspect to be considered.

Structural coverage. The simplest example of an objective testing criterion is that of structural coverage and, of such measures, statement coverage is the easiest to consider initially. In fact, since the scope of this study is not safety-related code, going beyond statement coverage to (say) branch coverage does not seem justified.

We now have a dilemma. Should the customer request 100% statement coverage by the supplier? This would appear to be a good idea, since it would provide an objective measure of testing which can be independently checked, if required. Unfortunately, obtaining 100% coverage is expensive in many cases, since the exact conditions required to execute a statement may be difficult to discover. In practice, a tool would be applied to discover the non-executed statements, and then either further tests written, or a justification produced as to why the statements are not executable (such as in the case of defensive programming). Undertaking this work is entirely appropriate for Level 3 and 4 software in the safety context [5], which would require statement coverage anyway.

100% statement coverage for less critical software is unlikely to be cost-effective, simply because the the most significant errors are likely to arise from a misunderstanding of the specification or 'missing code'. Showing that the code which is present does what the programmer intended is useful, but is almost certainly not the best use of resources.

What about less than 100% coverage? Conceptually, this seems less than ideal. Moreover, there is no good rationale for a value less than 100% (after allowing for defensive code). Unfortunately, there is a very wide variation in the effect of the percentage — for example, with data driven techniques 100% coverage may be easy to attain but means relatively little. In contrast, directly programmed data validation code can be very effectively tested by (100%) statement coverage.

The following comment was made to me, which I have confirmed:

> *Mike Hennell used to quote one example of a site where the staff wrote and tested their own code and had to hit some target (80%) for statement coverage — so they added dummy statements on the paths which were easy to test!*

We therefore conclude that direct specification by the customer of structural coverage metrics to be obtained during development is not appropriate.

7 A proposal

The author believes that producing high software quality requires hard work. In other words, there is no magic formula or silver bullet to obtain success. (In some highly specialised areas like producing a parser for a compiler, we do have tools which do the job, but they do not generalise.)

The problem faced here is to exploit objective testing as means of assurance. In doing this, we must avoid the *tick box* approach whereby some task is undertaken because *it is required* rather than because it directly contributes to the quality of the software.

Some success can be claimed in using both static and dynamic metrics in auditing the Channel Tunnel software [1, 18, 19]. This is important, since it appears that that software is typical of the military command and control systems which are of primary concern to the Ministry of Defence. Moreover, CSE International is accredited by UKAS for the application of these techniques, which demonstrates their objective nature [3].

It is unfortunate that no information is available at present on the (objective) testing to ISO 12119.

The proposals here cover functional and structural testing but avoiding the problems noted above. A sample response to these proposals and guidance is provided in the two appendices to this report.

Functional testing. We assume that the software is delivered with a 'user manual' which describes all the basic functionality of the system. (This might require some lateral thinking to gather together the information which could be used instead of a user-manual for those systems without such a document.) It should specify the effects of the user providing incorrect input, but need not provide any information on the affects of internally detected software errors.

On delivery of the software, the supplier should provide:

- A user-manual or equivalent describing all the basic functionality of the software.
- Examples of the functionality specified within the user-manual which can be demonstrated to perform as expected.

Structural testing. On delivery of the software, the supplier should state

- The number of components which were regarded as critical for which structural coverage data has been obtained during component test.
- The total number of statements and the percentage executed in the above components.
- The date of first issue of the critical components and a summary of the error statistics logged since then. The summary should indicate the time distribution of the errors and the percentage of those that are serious enough to require correction before a release of the main software.

For the complete system (perhaps without low-level device drivers so that the data can be obtained on a host environment):

- The total number of components in the system.
- The percentage of components executed at least once during system test.

COTS. The supplier should indicate the size of the object code of COTS components, and provide error statistics for these components.

Stress testing. The supplier should indicate how, if at all, the entire system has been subjected to stress testing, i.e. functional testing not based upon specific requirements, but aimed at combining requirements in an aggressive manner.

In all cases, the supplier should indicate any likely changes to the above statements with subsequent releases of the software.

Annex A gives an (artificial) example statement, and Annex B gives some guidance on what should be expected from such a statement.

7.1 Comments

Comments that have been made on a draft of this document but which have not been directly acted upon are collected here giving the reason for keeping the proposal in its current form:

Only SIL1 and SIL2 projects: Of course, the Ministry of Defence has many safety-critical projects. However, these have objective testing criteria build into their specification and hence could be excluded from the study.

No relationship with the CMM levels: This study seems to run along different lines than the SEI-CMM system and hence drawing comparisons did not seem worthwhile.

Add risk analysis: This would clearly be advantageous. There is no doubt that one has more confidence in projects which seem to be driven by the same view of risks that one has oneself. However, it seems too difficult to make this objective, as the commentator admitted.

One comment neatly summarises the problems: *I am convinced that we are at least a generation away from understanding how to "really" evaluate software in any truly meaningful, repeatable, objective manner.* The author interprets that statement to mean that not too much should be expected from this study.

8 Acknowledgements

Comments were made by the following on this study and these are gratefully acknowledged: Ted Aldous (GEC), Bruce Elliott (Praxis Critical Systems), Viv Hamilton (Safety Critical Systems Group, GMRC Gt Baddow), Debra Herrmann (CSC), C. Michael Holloway (NASA Langley Research Center), Peter Ladkin (Universität Bielefeld, Germany), Harry Lindsay (BAe SEMA), Chris Maund (GEC), Stuart Palin (Flight Systems Division, GEC Marconi Avionics Ltd), Mike Pickett (BAe SEMA), Professor Ian Pyle, Stuart Reid (RMCS), William Robertson (BDQ), Daniel Sniezek (Lockheed), Dr Victoria Stavridou (Visiting Scholar, Stanford University), Richard

Taylor (CSC Computer Sciences Ltd), Charles Waite (Data Refining, Inc.), Tony Wood (CSE International). I should add that there was a significant divergence of views on the comments, and hence there is no implication that any of the above support the results of the study in spite of my attempt to take all views into account.

References

[1] P A Bennett. Software Development for the Channel Tunnel: A summary. High Integrity Systems, Vol 1 No 2, 1994. pp213-220.

[2] British Computer Society Specialist Group in Software Testing. Standard for Software Component Testing (Working Draft 3.3). Glossary of terms used in software testing (Working Draft 6.2). April 1997. BSI draft for comment: references DC 97/644970 and DC 97/644971.

[3] UKAS Schedule of Accredited Testing. CSE International.

`http://www.cse-euro.demon.co.uk/cse_cert.htm#CSE_NAMAS`

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

[5] IEC 1508: Draft. Functional safety: safety-related systems. Parts 1-7. Draft for public comment, 1995. (Part 3 is concerned with software which is the relevant part for this study.)

[6] ISO/IEC Guide 25 1990. General requirements for the competence of calibration and testing laboratories.

[7] 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.

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

[9] "Information Technology Security Evaluation Criteria", Provisional Harmonised Criteria. Version 1.2. 1991. (UK contact point: CESG Room 2/0805, Fiddlers Green Lane, Cheltenham, Glos, GL52 5AJ.)

[10] 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.

[11] Defence Standard 00-42 (Part 2). Reliability and Maintainability Assurance Guides: Part 2: Software. 1st September 1997.

[12] Defence Standard 00-55, "The Procurement of Safety Critical Software in Defence Equipment", Ministry of Defence. 1st August 1997. Available free on the Internet:

`http://www.mod1ndrl.demon.co.uk/0055/0055.html`

[13] Chief of Defence Procurement: Instructions for Project Management. Defining Requirements in Software-Dependent Projects. CDPI/TECH/420, Issue 1.0, Paragraph 2.4. (To be placed on the Internet.)

[14] NAMAS Accreditation Standard. NAMAS. 1992.

[15] B A Wichmann. Testing Ada fixed point operations. NPL Report CISE 11/97. February 1997.

[16] B A Wichmann. Software in Scientific Instruments - Measurement Good Practice Guide No 5. May 1997.

[17] B A Wichmann. Some Remarks about Random Testing. To be published (available from the author).

[18] A P Wood. Experience and benefits of metrics use in verification activities. High Integrity Systems, Vol 1, No 5. 1996. pp485-493.

[19] A P Wood. Static and Dynamic Testing of the Channel Tunnel's Control and Communications Software. CSE International. 1997.

[20] Papers on SoftTest: What OVUM said about SofTest, Customer Highlights, Automated Requirements Based Testing (Bender and Associates). BDQ. 19 Woodend Drive, Ascot, SL5 9BD.

A Example statement from a supplier

This delivery of the XYZ Command and Control system is dated 29th February 1998 and the following statement on Objective Testing Criteria applies:

Functional testing.

- The user-manual issue 1.13 describes all the basic functionality of the software.
- All the examples in the user-manual have corresponding test scripts in our development system which are used to regression test every version, included those delivered.

Structural testing.

- The number of components which were regarded as critical for which structural coverage data has been obtained was 125.
- The total number of statements in these components was 135,456 and the percentage executed in the above components 92%. The 8% of the code which was not executed was concerned with handling more than one hardware error.
- Excluded from the above analysis was 20KBytes of code provided by suppliers A and B which implement the run-time system for the Ada language and the mathematical functions respectively. Since these products are available only in binary, we have no means of ensuring structural coverage.

The complete system without the low-level device drivers (which amount to 13,400 lines of code) was tested as a system on the host development environment from which the following statistics were obtained:

- The total number of components in the system was 15670.
- The percentage of components executed at least once during system test was 91%.

Note that the above tests have as a large subset the functional testing derived from the user-manual as specified below.

Stress testing. A trial has been conducted with 10 users attempting to stress-test the system. This trial was successful, but not objective enough to provide convincing evidence of the reliability of the software. We hope to provide a means of re-running stress-testing scripts to provide better evidence in the next release.

We do not see any likelihood of changing any of the 13,400 lines of critical components and hence the statistics there will remain unchanged.

We expect to raise the percentage coverage of the systems test by one or two percent, that is to 92% or 93%.

We are aware of inadequacies in the user-manual which implies that the functional testing could be improved. We await reactions from our user to decide which aspects should be enhanced.

B Guidance on the Supplier's Statement

The statement is repeated below but with guidance both for the supplier and for the Ministry.

Since about 85% of the problems with software are issues associated with the specification, correspondingly more effort should probably be spent addressing issues on functional testing rather than structural testing.

Functional testing. We assume that the software is delivered with a 'user manual' which describes all the basic functionality of the system. It should specify the effects of the user providing incorrect input, but need not provide any information on the affects of internally detected software errors.

It appears that the existence of a user-manual or a similar document is a reasonable assumption. However, it could easily happen that such a document does not exist, in which case, the other information requested may not be relevant.

Often functional testing is not based upon the user-manual but upon the original customer specification. Specification errors could then reveal themselves as inconsistencies between the software and the user-manual.

If software is to be maintained over a long period, then after the first 'complete' release and acceptance, it may be that the user-manual will be the key document rather than the original specification. If this is the case, then the proposal here that the testing is judged against the user-manual is a long-term benefit.

On delivery of the software, the supplier should provide:

With large systems, it is not practical to deliver everything at once (unless some items are deliberately delayed!). Hence it this context, we are concerned with a logical release, that is a consistent set of software with documentation.

- A user-manual or equivalent describing all the basic functionality of the software.

  It appears that all systems have such a user-manual or something which is more or less equivalent. However, the user-manual is often derived from the software specification or from actual use of the software, rather than the user requirements. This implies that inconsistencies could arise — perhaps the user needs to review this item separately from the software itself.

  The key issue to address here is the nature of the user-manual, highlighting the following:

  1. *To what extend does the user-manual cover all the essential functionality of the software?*
  2. *Has the user-manual been independently reviewed?*
  3. *Has the user-manual been reviewed by the customer?*
  4. *Has the user-manual been reviewed by a trained user of the system?*
  5. *How mature is the user-manual?*

In answering these objective testing questions, the above points should be taken into account, although it is not necessary to specifically address the five questions above.

- Examples of the functionality specified within the user-manual which can be demonstrated to perform as expected.

Structural testing. *The questions below assume that the system is layered into at least two levels: critical and non-critical. If such a layering is not present in the software, then it may be difficult to present a case that even quite modest reliability targets have been satisfied. Hence at this point, the rationale for the systems design and the manner in which reliability targets are being demonstrated should be presented, especially if the system contains no 'firewalls'.*

On delivery of the software, the supplier should state:

- The number of components which were regarded as critical for which structural coverage data has been obtained during component test.

  Clearly, this question implies that structural coverage data has been collected for the critical components. It is quite likely that this coverage data has been collected much earlier in the project. Only if the components have been changed would it be necessary to repeat such testing. A regression testing strategy would be appropriate here, since these (low-level?) components should be more stable than the higher-level ones.

  As noted in the main report, the customer should not *specify coverage target in advance. Nevertheless the supplier should understand what targets would be the best compromise, taking into account the coding style and nature of the application-code.*

- The total number of statements and the percentage executed in the above components.

  If a high figure is provided (say 90%), then this may not be such good news, since it is possible that it indicates a specific coding style, or absence of defensive programming, rather than high quality software.

  Figures lower than around 70% should probably be justified. Extensive use of defensive programming could be a cause. Another cause can be logical complexity not directly related to the input so that the unexecuted statements cannot be traced back the specific input. (The code generation phase of a compiler is like this.)

- The date of first issue of the critical components and a summary of the error statistics logged since then. The summary should indicate the time distribution of the errors and the percentage of those that are serious enough to require correction before a release of the main software.

  This question implies that the reliability target are being met, at least in part, by ensuring that the critical components are robust. Again, if such layering is not part of the design, then any alternative needs to be documented.

  Useful reliability data for the critical components can probably only be presented if the critical components are relatively stable and have demonstrably reducing error report frequencies.

For the complete system (perhaps without low-level device drivers so that the data can be obtained on a host environment):

The implication here is that host testing is effective. If this is not the case, then the reasons should be stated. Host environment testing should be possible with minimal manual intervention (i.e. regression testing).

If a large system is being provided in several phases with a large increment in functionality for each one, then regression testing may not be so effective, due to the change in the specification for each phase.

- The total number of components in the system.
  Obviously, the smaller the number of components, the easier it should be to gain assurance from testing or fault report history.
- The percentage of components executed at least once during system test.
  This question implies that monitoring of the execution is possible when undertaking the system test. This could be on a host system, or the target.

COTS. The supplier should indicate the size of the object code of COTS components, and provide error statistics for these components.

If there is a large COTS component, then the question arises as to how assurance can be gained from this. COTS does not, of itself, imply any specific reliability or quality. The size of the user-base could be useful, as could an indication of the maturity of the product (assuming data from real use can be obtained).

Stress testing. The supplier should indicate how, if at all, the entire system has been subjected to stress testing, i.e. functional testing not based upon specific requirements, but aimed at combining requirements in an aggressive manner.

Using first-time users to try to break the system is helpful, but not objective and highly dependent upon the skill of the user. Programs or scripts that produce reproducible tests cases are better, but require skill to produce effective testing evidence. The testing should encompass all the basic functionality, but combined in an aggressive fashion. For details of this technology, see [17].

In all cases, the supplier should indicate any likely changes to the above statements with subsequent releases of the software.

In some cases, a system is delivered in a number of phases, with very substantial additions in subsequent phases. In these cases, data from the reliability of previous phases is not necessarily indicative of the next phase. On the other hand, if a release is merely supposed to correct reported faults (i.e. no change in the specification), then an improvement in the reliability should be assured.