Quality Assurance of Medical Software

P.S. Cosgriff
Medical Physics Department, Pilgrim Hospital, Boston, UK

Abstract

Quality assurance is a topical issue within the software industry and various methodological solutions aimed at improving software quality have been proposed. Medical software is generally deemed safety-critical and, as such, should be subject to control procedures exceeding those applicable to general purpose software. The user/purchaser of medical equipment has an important part to play in the drive for higher quality medical software, and ways in which the user can influence commercial suppliers are outlined. The implications of existing and planned legislation are also discussed.
1. Introduction

In this paper the term 'medical software' refers to the software component of a medical device used for diagnosis or treatment of disease. Most electro-medical equipment now contains software for control and/or data processing purposes. If the device is not programmable by the user the software is said to be 'embedded'; usually stored in read-only-memory. Typical examples would include modern electrocardiography machines, infusion pumps, and most intensive care equipment. In such cases, the ward user may be unaware that the device is computer-controlled. Software is used in a more obvious way in radiographic equipment for image processing, where system access is usually via the familiar keyboard/monitor arrangement.

However complex the equipment, the clinician probably has a simple view of it; understandably expecting it to perform the task for which it was purchased. The realisation that computer hardware and, in particular, software is not as reliable as was assumed can be a sobering experience [1].

The main reasons for poor quality medical software are the user's historical failure/inability to properly specify the software requirements of the medical device, and the lack of adequate software quality assurance (SQA) procedures employed by manufacturers [2]. In addition to QA requirements arising from general concerns about functionality, usability, etc, the safety-critical nature of medical software places extra responsibilities on developers.

Product quality in any context is usually defined in terms of 'fitness for purpose', so it is self evident that the user is the arbiter of quality. Software products are either made to a particular customer's order (bespoke or custom software) or are produced with a general market in mind (packaged software). Hospitals employing staff with the requisite information technology (IT) skills may decide to undertake medical software development projects in-house. This is a special case of bespoke development for which the roles of the individuals involved need to be very carefully defined. SQA methods prescribe active user involvement in the development process, particularly in the requirements analysis and system design stages, and may be readily applied to bespoke development. For procurement of packaged software, the process is essentially one of obtaining the 'closest fit' to the specification.

Medical software is generally packaged (i.e. the medical user purchases a finished product), so the discipline of conventional SQA techniques cannot be fully applied. However, by virtue of the close relationship that exists between many medical device manufacturers and their customers it is possible to approach the 'custom-built' technique. For all substantial software procurement projects, it is essential to produce a detailed specification. Although this sounds an obvious statement, inadequate specification is usually the root cause of failed software projects.

In this article the role of the user is generally taken to be that of purchaser (rather than developer), with emphasis on recommending methods that the user can reasonably expect the supplier to use during the development process. It is assumed that the user requires a medium-to-large software system whose price would justify the costs involved (senior staff
time) in preparing a comprehensive specification, inviting demonstrations, organising tenders, etc. Budgets for requirements analysis are typically set at 5-10% of the envisaged capital cost [3]. A example of a medium sized medical software system would be a suite of isodose plotting programs to aid radiotherapy planning. Because of its specialised nature, however, all commercial medical software is relatively expensive (c.f. business software), so it rarely makes sense to cut corners in procurement.

One of the main problems encountered by personnel new to the subject of SQA is forming an understanding how the various elements fit together (and overlap). Although there are many different guidelines, standards and methods, they generally fall into one of two categories: software project management or software engineering.

The general approach taken in this paper is to use the international software quality management standard ISO 9000-3 [4] (see section 2.2.1) as the basic framework, and to indicate methods which are useful in supporting it. The structure of ISO 9000-3 is shown in appendix 1. Issues specific to medical software (e.g. safety-criticality and legislation) are considered separately.

Software development projects need to be approached in a systematic fashion, which implies the use of a model (referred to as a life-cycle model) to act as a framework for the whole project - a basic requirement of ISO 9000-3. The software development life cycle (SDLC) used to illustrate points raised in this paper is shown in figure 1.

2. Software quality management systems

2.1 General

Software quality management systems (QMS) are concerned with processes. The international standard for quality management systems is ISO 9001 (BS5750:Pt1). Associated third-party accreditation schemes provide a means by which the manufacturer can demonstrate a commitment to quality (i.e. ensure recognition). It is emphasised that an ISO 9001 audit is not concerned with testing the product.

Software QMS standards relevant to medical devices include those of a general nature, those concerned with safety-critical systems, and those specific to medical devices. It has been successfully argued that the general quality systems standard ISO 9001 is not directly applicable to the development of software, and a specific guideline standard (ISO 9000-3) has been produced. It is possible for a manufacturer to gain specific software development accreditation through the 'TickIT' scheme (run by the U.K Department of Trade and Industry, DTI), which is concerned with third party certification of software quality management systems. TickIT auditors inspect with reference to ISO 9001, and use ISO 9000-3 as the applicability guide. ISO 9000-3 is simply a set of requirements which can be related to a life cycle model.

The TickIT guide to software quality management systems is available [5] which includes a full copy of ISO 9000-3, as well as guidance for both purchaser and supplier. The suppliers' guide includes a description of one particular quality management system (QMS) distilled from
a wide variety of IT environments. It is, in effect, an expanded version of ISO 9000-3, using a standard format to discuss each aspect (activity definition, objectives and criteria, outputs, standards and procedures, control mechanisms, approvals, comments, and ISO 9001 reference). The TickIT guide provides more detailed information on what should be done, but not how to do it. It is expected that "...developers will use a variety of methods to implement the principles of the guide, depending on size and complexity of the software project in hand". The developer must therefore choose a suitable methodology for each phase of the development process (see section 3.2).

The purchasers' guide outlines what users can expect of a suppliers' QMS, paying particular attention to the purchaser-supplier relationship. Issues are viewed in contractual terms. The purchasers' QA responsibilities may be summarised under the headings: invitation to tender, in-contract activities (design and development), end of contract (delivery), and post-contract (maintenance). There is also a need for the purchaser to monitor the suppliers' QA activities during the development stages. The guide indicates points at which explicit user approval is required for the project to proceed (e.g. formal agreement of the requirement specification) and points at which the purchaser should satisfy himself that the developer is indeed monitoring the development process in accordance with the project quality plan.

Most of the user's input to the development (or procurement) process is focused on two phases of the SDLC. The first, already discussed, is the requirements specification. The second is acceptance testing. It is essential that acceptance criteria are established during the user requirements phase, so that the developer is given fair warning of the aspects of the system which will be specifically tested by the user. The method of handling problems discovered during the acceptance procedure should also be agreed (in writing) between purchaser and supplier in advance - as indicated in ISO 9000-3 (paragraph 5.8.1).

Individual utility programs can be tested using synthetic test data [6] and clinical software (e.g. software for calculation of some physiological parameter) may be tested by use of real patient data. For the latter, the test objects are referred to as software phantoms, and libraries of validated patient data are being established for software QA purposes in nuclear medicine [1] and electrocardiography [7]. In principle, software phantoms can be distributed to users (e.g. via floppy disk) to aid in acceptance testing and subsequent quality assurance [1]. There would also appear to be a role for professional bodies in maintaining such external QA programmes once they have been established by individual research groups.

ISO 9000-3 specifies (paragraph 5.4.4.2) that the overall development plan should define how the project is to be managed, including the identification of organisational responsibilities, resources and schedules. This would include, for example, information on who is to conduct the in-process testing (e.g. code inspections) and phase reviews which are at the heart of software quality assurance. For obvious reasons, these 'internal audit' functions should be performed by specifically trained staff (within the supplier's team) not directly involved with the production of the software. The PRINCE project management method [8] is a useful guide to the management of IT projects and can be used to support the development planning aspect of ISO 9000-3 (see appendix 1).
In summary, the key points of ISO 9000-3 are that:

- It is based on the process of software development (i.e. refers to planned phases as part of a life-cycle)
- It recognises the importance of user involvement, and indicates how the input should be organised.
- It makes testing, verification / validation, configuration management and quality measurement mandatory.

The discussion so far has largely concerned process issues. In the absence of standards detailing product requirements, it is difficult to establish a relationship between conformance to the (process) standard and quality of the end-product. One of the fundamental problems in the field of software development is the lack of scientific assessment for process standards themselves, and the tools and techniques to which they refer. It has even been argued that, by definition, software standards are not standards at all [9]. The methods and techniques referred to in section 3 have a wide body of support on the basis of the thorough approach which they prescribe, but it has to be admitted that there is little hard scientific evidence of their real benefits.

It is nonetheless recommended that purchasers of medical equipment specify compliance with both ISO 90001 and ISO 9000-3 in tender documents relating to the intended purchase of software-containing medical devices. Suppliers holding a TickIT certificate would effectively fulfil both requirements. By this simple action users will speed up the adoption of these standards by manufacturers. After 1997, the requirements may well become mandatory (see section 4.1).

2.2 Safety-critical systems

The purpose of this section is to introduce some basic issues in the design of safety-related systems, and to provide references for interested readers. The term safety-critical is usually reserved to describe systems whose failure could result in absolute harm (i.e. irremediable or irrecoverable damage) to individuals or the environment. Examples of safety-critical systems would thus include an aircraft in flight, a nuclear power station or a patient connected to a life support machine. Although use of the term safety-related is sometimes used to describe a system whose failure would have less serious consequences, it is here used as a generic term covering all systems having safety implications. Also, in this paper, the term should be read as software-based safety-related system (SRS). The term Programmable Electronic System (PES) is used to cover all types of computers from single microprocessors to fully integrated systems with high level languages.

A computer, and hence software, can only affect safety if it is used to control or influence a process which can lead to physical harm. The International Electrotechnical Commission (IEC) uses the term Equipment Under Control (EUC) to describe the host device. Although medical equipment is generally referred to as safety-critical, this is only strictly true for certain types. Most, however, would fall into the broader 'safety-related' category. Clearly, a software design fault in a drug infusion pump is potentially much more dangerous than a 'bug' in a diagnostic
device, where there would always be some 'competent human intervention' [10] between the output of the computer and any consequential treatment of the patient.

The safety-criticality of a particular piece of equipment depends entirely on the context in which it is used. For example, the malfunction of a (microprocessor-controlled) syringe pump used in a laboratory experiment may have no safety implication, but the same malfunction in the same device used to infuse a potent drug into a patient could be extremely dangerous.

In a clinical environment, the risks of software failure (i.e. equipment failure caused by activation of a software fault) must be balanced against the benefits of using a particular medical device. The benefits to the patient of using the device will generally outweigh the small risk of software failure, although this is no reason for complacency on the part of medical software designers.

For computer-controlled equipment, it is important distinguish between safety and software quality. Software quality assurance is obviously important and - through its emphasis on defect prevention - has a bearing on safety, but improving software quality does not necessarily lead to improved safety*. Software may be correct, but unsafe, due to errors in the specification. Similarly, the system may be incorrect (i.e. fail to implement some aspects of the specification) and unreliable, but safe because its failure modes are not hazardous. Unlike software quality, safety is a systems issue, which means that the whole operating environment - including human interactions - need to be taken into account in the development process. Training of human operatives is thus a major safety issue. For example, safety-protection software built into a patient ventilator may correctly detect the presence of a particular fault, but unless the medical user (e.g. anaesthetist) has been properly trained to recognise the fault signal, and shown what to do about it, the system will be unsafe.

Although the use of computers in safety-critical equipment offers potential advantages, these will only be realised if appropriate methodologies are used in the development process. Software allows highly complex systems to be built and, since complexity is one of the main 'enemies of safety' [11], strict adherence to well documented methods is therefore particularly important in this context. An obvious way in which a PES might improve safety would be to undertake certain monitoring tasks which human operators perform notoriously badly - a process known as 'shifting the man-machine partition'.

In engineering generally, absolute safety is an unobtainable goal and designers of safety-critical systems use the concept of 'tolerable risk' to determine a reasonable end-point for their efforts. Risk is here defined as "..the 'product' of the consequence of a hazardous event and the frequency, or probability, of its occurrence" [12]. Safety can thus be improved by reducing the effect of the hazardous event, or reducing its frequency. Setting a tolerable risk level may well involve consideration of social and political factors, as well as technical ones. Once the tolerable risk has been established, the designer can allocate the system safety integrity level, which is the probability of the SRS satisfactorily performing its designated functions.

* Use of term 'software safety' is to be discouraged since it wrongly implies that the software can be considered in isolation.
As with quality requirements such as security, expandability, etc, safety has to be designed-in from the outset, which implies that safety requirements be explicitly stated in the specification. Furthermore, as many safety requirements as possible should be translated into measurable functional requirements. These requirements would then lead the developer to design and implement specific safety features whose effectiveness could, in principle, be measured.

Such features may well include a so-called 'software protection system' (SPS), which monitors certain aspects of the main control software and, in the event of malfunction, takes appropriate steps to ensure that the system does not enter an unsafe mode. The essential characteristic of the SPS is that it is separate to the control system being monitored, thus offering an added layer of safety assurance. The SPS thus improves safety by reducing the probability of a hazardous event. The ease with which the system can be returned to a safe mode obviously depends on the context. The system is said to be fail safe if there is a safe state that can be reached easily and with high reliability. In some circumstances simple 'graceful' shutdown may render the equipment safe. In other circumstances (e.g. an aircraft in flight, or a patient ventilator) it certainly wouldn't. Where systems need to continue functioning for some time after the fault has occurred the term fail operational is used.

A concept used extensively in safety-critical systems is redundancy, which will usually apply to both sensors and software. Dual software systems running on separate hardware in parallel provides a means of checking for errors (action taken only if both systems agree) as well as allowing the EUC to continue functioning if one of the control systems fails. Dual systems therefore improve safety by both reducing the frequency of hazardous event and reducing the effects of failure. In the medical domain such complex and expensive systems may be reserved for treatment devices with potential hazardous mechanical components, such as surgical robots [13].

Two draft generic standards for the development of systems for safety-related applications (e.g. nuclear power, civil aviation, health care) have recently been produced by the IEC. One [14] relates to systems issues while the other [15] relates to specific software issues. The systems standard proposes a model (overall safety life cycle) to establish a link between the risk assessment of the EUC and the safety integrity of the SRS. There is currently no specific standard for medical software, although this is currently being considered by working group 6 of CEN (Comité Européen de Normalisation) Technical Committee 251 (Medical informatics) and working group 2 of IEC Technical Committee 62 (Electrical equipment in medical practice). Progress is slow, however, and it may be some time before agreed standards emerge.

There are a number of organisations aiming to raise awareness of software safety issues amongst users and manufacturers. In the UK there is an Interdepartmental Committee on Software Engineering (with representatives from the DTI, Department of Transport, Department of Health, Ministry of Defence, and the nuclear power industry) which has a working group considering safety-related software. The DTI and the Science and Engineering Research Council (SERC) support the Safety-Critical Systems Research Programme and,
together with other government departments and organisations, launched the 'SafeIT' initiative [16,17] from which the Safety-Critical Systems Club* emerged.

In summary, the key to designing safety-related systems is to critically assess the potential safety risks (at the design stage) and to take appropriate steps to minimise both the frequency and effect of failure. If successful, the safety of the system will not then depend arbitrarily on software quality. Risk analysis will identify 'hot spots' within the system requiring special attention in the design. The 'special attention' should be detailed in a safety plan [11], which is analogous to the quality plan required for general software quality assurance. The safety plan should include reference to specific techniques, tools and methods which are to be used to address the safety issues. The use of so-called formal methods have been advocated for the verification steps [18], although they are not in widespread use for reasons of practicality [19].

As will be obvious from this brief discussion, the development of safety-related systems is a complex issue, requiring the combination of two distinct disciplines: software engineering and safety engineering. Software developers undertaking projects involving safety related equipment should use currently available (process-orientated) methodologies and standards (e.g. ISO 9000-3), plus appropriate safety measures, some of which have been referred to above. Specific product-orientated tools and techniques are available for safety-related development but as yet there is no consensus on 'best practice'.

3. Software engineering

As previously indicated, software quality management systems (e.g. ISO 9000-3) act to constrain and control the development process. The least that can be expected following introduction of a recognised QMS is an improvement in consistency but, if this is the only benefit, the quality of the end product will not necessarily improve. What is missing are the methods and techniques to control product quality directly, which leads to consideration of software engineering (SE) issues. The three main components of software engineering are models, methods and measurement which, if used within a QMS environment, can make a significant contribution to software quality assurance.

3.1 Models

Life-cycle models have always been the basis of software engineering and have already been mentioned in a QA context. They are especially useful because they provide a common framework for both project and quality planning. The V life-cycle model (figure 1) not only shows the sequence of activities, but also helps identify which verification and validation procedures are appropriate to each phase of the development cycle. Being diagrammatic it is also readily understood by the user. The end of each life cycle phase is an obvious quality review point, but there will be others which emerge from detailed planning.

---

* For more information about the Safety-Critical Systems Club, contact the Centre for Software Engineering, Computing laboratory, 20 Windsor Terrace, University of Newcastle, NE1 7RU.
3.2 Methods

Software engineering methods relate to phases of the SDLC and are termed development methodologies. By comparison, a QMS such as ISO 9000-3 may be regarded as a control method. A development methodology is essentially a refinement of the life cycle model into a specific process.

The idea of a methodology is to define the SDLC itself (in terms of its phases), the deliverables to be produced during each stage, and the verification and validation procedures appropriate to each stage. Several methodologies have been published covering different parts of the life cycle. They are generally highly structured and prescriptive. SSADM, the Structured Systems Analysis and Design Method [20], has been in existence since 1980 and has become popular within the UK for guiding the requirements analysis and design phases of the project. The strength of SSADM is that it is diagram-based, providing for good communication between members of the development team and, equally importantly, between the developer and the user representative(s). The method is promoted by the Central Computer and Telecommunications Agency (CCTA) - who co-developed it - and the National Computing Centre. Control of SSADM has recently been passed to the British Standards Institute.

There are, however, some problems with SSADM. First, it is a relatively complex method with numerous stages, some of which are regarded as non-essential by some developers [21]. Perhaps more importantly it only covers the stages of the project up to and including the design phase, so another method has to be employed for the remaining phases - which presents a dovetailing problem. ISO 9000-3 stipulates (paragraph 4.2.1) that "..the quality system should be an integrated process throughout the entire life cycle". The advent of specific CASE (computer aided software engineering) tools has made the adoption of structured methods like SSADM more of a practical proposition [22].

3.3 Measurement

Since measurement is the essence of quality assurance, there is general agreement that so-called metrics have a key role to play in SQA, provided that the ones chosen reflect the problems of the process or product being measured [23]. A reliable metric is invaluable to a project manager since it allows objective targets to be set for various tasks.

3.3.2 Process metrics

Quality factors implied in user requirements documents may include usability, portability, reliability, expandability and security. The nature of the system will often dictate the prioritising of these desirable features. A typical mapping of system characteristics to software quality factors is shown in table 1. A developer may reasonably be expected to incorporate up to four (top-priority) quality factors in the design; specifying more means that the project becomes too complex. This does not necessarily mean that other factors will be absent, although it should be appreciated that some factors tend to conflict (e.g. security and efficiency).
The way in which the developer attempts to ensure that the required quality features are built into the software is by reference to a software QA model (not to be confused with the life-cycle model), which links a given quality factor (e.g. expandability) to measurable software characteristics ('metrics'), via a set of software attributes (fig. 2). It should be noted that the meaning of terms such as reliability, portability, etc, within a software QA model will generally be somewhat different to their lay definitions and it is therefore essential that user and developer agree on the terminology. The essential feature of such models is that the appropriate metrics for a given quality factor are prescribed. Individual metrics are grouped into metric score sheets and the project manager sets an appropriate score for a given phase activity as a target. Details of an SQA model developed by the Boeing Corporation have been published elsewhere [24].

It should be appreciated that the use of such models and methods will increase the 'QA overhead' considerably, and should perhaps be reserved for systems which are truly safety-critical (see section 2.2). Even without an in-process metrics programme the user can still specify features such as portability, expandability, etc, but the validation will be limited to testing for the existence of the feature in the 'final' product, which could be at system test and/or acceptance test stages. Clearly, the quality requirements must be specified as measurable targets for this type of validation. For portability, this might mean a sub-set of system tests being performed on different versions of the software - each designed to run on a different (specified) hardware platform. ISO 9000-3 implies (paragraph 6.4.1) that, as a minimum, metrics should be used to reflect the number and severity of field test / acceptance defects reported by the customer.

The simplest (and possible most effective) metric to use is the defect count. A defect is defined very broadly and may include everything from ambiguities in documentation to straightforward programming (coding) and logic errors. A defect report is produced for each fault detected which includes information on defect type, how it was detected, when it was detected (date / life cycle phase), and what decision was made (by the project manager) about its correction. A defects database is thus built up which not only helps with the project in hand but is also invaluable in planning improvements for future projects. This approach embodies the classic QA cycle which applies to almost any manufacturing process: collect defect data, analyse data, engineer process improvement.

4. Legislation

Standards are essentially voluntary, their adoption by manufacturers generally depending on the perceived benefits (by both manufacturers and users), and peer pressure. However, if a standard is referred to in legislation it obviously takes on a quasi-obligatory status [25]. This is currently the case for medical software sold in the U.S. and Europe is set to follow suit. It has largely been the occurrence of accidents leading to death or injury which have prompted governments to legislate. The best publicised case of catastrophic software failure in a medical device was the 'Therac 25 incident' in Canada in 1986, when two patients died and several more were injured after receiving overdoses of radiation from a Therac 25 radiotherapy machine [26].
4.1 EC medical devices directives

The need for medical device purchasers to investigate standards compliance issues with potential suppliers should be largely obviated when the forthcoming EC Medical Device Directives come into force. ISO 9001 accreditation will then effectively become a pre-requisite for the marketing of an electro-medical device - a fundamental change (within the U.K) to the current voluntary code of practice (Manufacturers Registration Scheme) run by the Medical Devices Directorate within Department of Health.

There are in fact three Directives with the following implementation dates:

1. Active Implantable Medical Devices - Came into force 1 January 1993
   - Transitional period to 1 January 1995

2. Medical Devices - Comes into force 30th June 1994
   - Transitional period to 30th June 1997

3. In-Vitro Diagnostic Systems - Comes into force 1 January 1996 (?)
   - Transitional period to 1 January 2000 (?)

The 'transitional period' is the period during which governments in member states must produce national legislation in support of the Directives.

Risk-based classification of medical equipment is central to the EC Medical Device Directives, since it provides the means by which appropriate conformance assessment procedures may be established for a particular device. The second EC Directive ('Medical Devices') uses a four-level classification system based on risk assessment [24,25]. Medical devices containing software will be classified as either IIb (intermediate risk) or III (high risk). In either case, conformance assessment (ref: ISO 9001 and possibly ISO 9000-3) for both design and manufacture must be performed by an authorised third party (to be known as a Notified Body).

When national legislation emanating from the Directives is in place, the situation in Europe will be brought into line with that in the United States, where the Safe Medical Devices Act 1990 embodies the preventative approach taken by the U.S Food and Drug Administration (FDA) since the late 1970's.

The EC Directives are expressed in terms of Essential Requirements (ER); the ER relating to contained software (ER 12.1 within the second Directive) stating that 'Devices dependant on software must be designed in such a way as to minimise the risks arising from errors in the program' - which is being interpreted as indicating compliance with ISO 9000-3.

It is uncertain at present how the Directives will affect in-house development of medical equipment, including software. However, even if in-house (non-commercial) developers are not forced to comply with the legislation, it would certainly be desirable from a professional point of view. Even if not caught by legislation emanating from the medical device Directives, activities of in-house manufacturers are certainly covered by consumer protection legislation.
4.2 Product liability

Part 1 of the Consumer Protection Act 1987 implements the 1985 EC Directive on product liability. From the wide definition of a product, it appears that computer software is covered by the Act [27] and producers would be well advised to assume this. The Act states that the 'producer' is liable for defects in the product (leading to personal injury) which, for software, would be the developer. The person suffering damage has to prove that it was caused by the defect but does not have to prove negligence on the part of the producer - a concept known as 'strict liability'.

The provisions of the Act clearly apply to in-house developers, as well as commercial manufacturers. Health circular HN(88)3 states that, for all manufacturing activities within the NHS, "...it is important that every effort be made to avoid producing defective products. This means that products should conform to any relevant standards which exist and that records are kept to demonstrate conformity". The guidance goes on to add that standards compliance does not provide a defence under the Consumer Protection Act (because faulty products can still occur) but will reduce the likelihood of claims and will increase the confidence of in-house manufacturing. This guidance is certainly relevant to in-house production of medical software. It could also be argued that ISO 9000-3 is a relevant standard.

If software is misused despite a clear indication by the manufacturer regarding potential application of the product, then the manufacturer may be able to shift liability to the user (clinician or other health care professional). In the U.S, where liability claims in the medical field have become large and numerous, much importance has been attached to 'warning notices' a manufacturer may have issued regarding correct and incorrect use of the product. Blanket disclaimers, however, can be ignored. They do not indemnify the producer.

5. Discussion

Specifying quality concerns is crucial to procurement of both custom and packaged software; the obvious difference being that a supplier of packaged software will simply state (in the tender documents) whether or not a particular feature is present in the product. It is therefore important to clearly define what is meant by terms such as 'security', for example, in order than the presence of such features can be readily verified by the customer prior to purchase (i.e. by demonstration and/or at acceptance testing). The same argument applies to the functionality, although this is usually a less of a problem because users tend to be more specific when detailing operational requirements. The way in which the functionality part of the software requirements fits within a specification for a medical device can be seen from recently produced guidelines for gamma camera tendering [6].

A common failing in writing medical software specifications has been to take too much account of what medical equipment manufacturers can currently supply. If the specification is pitched too low, nearly all the manufacturers will be able to meet it, making it difficult to justify rejection of systems which are known instinctively to be inadequate. Indeed, it would be very surprising if any packaged software vendor could meet all the requirements. Clearly, the system which comes closest would be chosen, but the manufacturers (including the one with the winning tender) should have been alerted to the presence of unmet specifications. If
enough users specify similar basic requirements - and there is no reason why practitioners in a
given medical specialty shouldn't - the company to improve its system in line with these
requirements would obviously gain a competitive advantage, leading to elimination of
products/manufacturers unable to meet the specification.

Some authors have advocated an even more rigorous approach to the procurement of
packaged software, based on a formal Request for Proposal (RFP) [3], but this is considered
justifiable for only very large software projects.

As implied, collective action by users is the key to success since manufactures are unlikely to
respond to the pleas, verbal or written, of individual users unless they are convinced that the
proposed new feature will appeal to the majority of other users; an argument which leads
naturally to the issue of standardization of user requirements. Through European collaborative
research, a document outlining basic user requirements for nuclear medicine information
systems (hardware and software) has recently been published [28] which represents an attempt
to influence the future general direction of software development within that specialty. Many
of the requirements will be common to other diagnostic imaging departments.

With increasing emphasis now being placed on medical and clinical audit, standardization of
individual clinical procedures (e.g. diagnostic tests) is now seen as desirable, and specific
proposals are starting to appear in the literature. If adopted they should, in turn, lead to
incorporation of specific software features required to support these standards. For example, a
proposed standard for gamma camera renography [29] contains a requirement that the
acquired data be displayed as a curve in real-time - a relatively simple software function
currently absent from most commercial nuclear medicine computer systems.

It is expected that over the next few years software development will become more user-
centred. Although many companies have talked about user-driven design for a number of years
this is only just becoming a reality. Encouragingly, one multi-national medical equipment
manufacturer has recently organised a series of formal user-centred meetings in Europe and in
the US to elicit system requirements from selected small groups of representative users. Most
importantly, the company has committed itself to basing the design of future products on the
(high priority) user requirements so produced. In the IT industry generally similar ideas lie
behind the Rapid Application Development (RAD) method [30] which is expected to spread as
manufactures attempt to speed up the development process - hence reduce costs - without
sacrificing quality.

On the safety front, formal methods will no doubt become more automated, and therefore
more acceptable. With further development, Natural Language Processors may well provide a
reliable means of extracting meaning from user requirements written in ordinary English,
thereby enabling a formal (‘mathematical’) link to be established between specification and
design.

The most important developments, however, are likely to involve people rather than technology.
Raised awareness of quality / safety issues amongst users and developers of IT, and better
training for project managers and software engineers should be the main priorities. In safety-
related areas such as medicine, software projects should only be undertaken by production teams
with the resources and skills to fully implement the existing / impending standards, which will inevitably mean less in-house development work in future.
Appendix 1  The structure of ISO 9000-3

Parts of some development and project management methods can be used to help comply with parts of ISO 9000-3. The references below to SSADM (for analysis and design aspects) and PRINCE (for project management) include, where necessary, the relevant sections headings within those publications.

1. Scope
2. Normative references
3. Definitions

4. Quality system - Framework
   4.1 Management responsibility
      4.1.1 Supplier's management responsibility
         4.1.1.1 Quality policy
         4.1.1.2 Organisation
            4.1.1.2.1 Responsibility and authority
            4.1.1.2.2 Verification resources and personnel
            4.1.1.2.3 Management representative
         4.1.1.3 Management review
      4.1.2 Purchaser's management responsibility
      4.1.3 Joint review
   4.2 Quality system
      4.2.1 General
      4.2.2 Quality system documentation
      4.2.3 Quality plan
   4.3 Internal quality system audits
   4.4 Corrective action

5. Quality system - Life cycle activities

   5.1 General
   5.2 Contract review
      5.2.1 General
      5.2.2 Contract items on quality
   5.3 Purchaser's requirements specification
      5.3.1 General .......................... [20; requirements catalogue]
      5.3.2 Mutual co-operation
   5.4 Development planning
      5.4.1 General
      5.4.2 Development plan
         5.4.2.1 Phases
         5.4.2.2 Management .................. [8; project plan, resource plan, technical plan]
      5.4.2.3 Development methods and tools
   5.4.3 Progress control
   5.4.4 Input to development phases
5.4.5 Output from development phases
5.4.6 Verification of each phase

5.5 Quality planning
  5.5.1 General
  5.5.2 Quality plan content .................... [8]

5.6 Design and implementation
  5.6.1 General
  5.6.2 Design .................................... [20; data flow diagrams, entity diagrams]
  5.6.3 Implementation
  5.6.4 Reviews

5.7 Testing and validation
  5.7.1 General
  5.7.2 Test planning
  5.7.3 Testing
  5.7.4 Validation
  5.7.5 Field testing

5.8 Acceptance
  5.8.1 General
  5.8.2 Acceptance test planning

5.9 Replication, delivery and installation
  5.9.1 Replication
  5.9.2 Delivery
  5.9.3 Installation

5.10 Maintenance
  5.10.1 General
  5.10.2 Maintenance plan
  5.10.3 Identification of the initial status of the product
  5.10.4 Support organisation
  5.10.5 Types of maintenance activities
  5.10.6 Maintenance record and reports
  5.10.7 Release procedures

6. Quality system - Supporting activities

6.1 Configuration management
  6.1.1 General
  6.1.2 Configuration management plan .. [8]
  6.1.3 Configuration management activities
    6.1.3.1 Configuration identification and traceability
    6.1.3.2 Change control
    6.1.3.3 Configuration status report

6.2 Document control
  6.2.1 General
  6.2.2 Types of documents
  6.2.3 Document approval and issue
  6.2.4 Document changes

6.3 Quality records

6.4 Measurement
6.4.1 Product measurement
6.4.2 Process measurement
6.5 Rules, practices and conventions
6.6 Tools and techniques
6.7 Purchasing
   6.7.1 General
   6.7.2 Assessment of sub-contractors
   6.7.3 Validation of purchased products
6.8 Included software product
6.9 Training
References


Figure 1. The main phases of the 'V' software development life-cycle (SDLC). The dotted lines indicate validation links between phases. Units are self-contained modules of code which are designed in such a way that they can be individually tested. The tested units are then assembled ('integrated') into a complete system. The phase boundaries are obvious major review points. Before proceeding to the next stage, the project manager should get signed authorisation from the nominated user representative. The 'maintain / enhance' phase is optional, covering the fixing of 'bugs' discovered by users after release, and general (unspecified) improvements that may be required in future. In a bespoke development application, these matters should be written into the contract. Traditionally, most user activity is concerned with phases 1 and 8.
Figure 2. The basic element of a metric-based model for implementing software quality assurance. Appropriate metrics for each 'quality factor' (e.g. usability) will be prescribed by the model. A numerical value (metric score) for a given activity is then set by the project manager to be consistent with attaining the user-defined quality target (e.g. that non-technical users should be able to find and master defined basic functions after 20 minutes of personal tuition by a member of the approved support staff and 1 hour of self use (user manual available). Within the Boeing model [24], the (defined) attributes for usability are 'operability' and 'training'.
Table 1. Mapping of some system characteristics to associated quality factors

<table>
<thead>
<tr>
<th>SYSTEM CHARACTERISTIC</th>
<th>QUALITY FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety-critical</td>
<td>Reliability</td>
</tr>
<tr>
<td></td>
<td>Correctness</td>
</tr>
<tr>
<td></td>
<td>Verifiability</td>
</tr>
<tr>
<td>Classified data</td>
<td>Security</td>
</tr>
<tr>
<td>Real-time operation</td>
<td>Efficiency</td>
</tr>
<tr>
<td>Different operating systems</td>
<td>Portability</td>
</tr>
<tr>
<td>Non-technical users</td>
<td>Usability</td>
</tr>
<tr>
<td>Further developments planned</td>
<td>Expandability</td>
</tr>
</tbody>
</table>