QA Training Materials

Software Testing and Quality Assurance

The stated learning AIM is listed as follows …

- To instruct in applied Software Testing and associated Quality Assurance including best practices and industry standards.

The stated learning OBJECTIVES are listed as follows …

- It is the stated learning objective to introduce and demonstrate the practicalities and technicalities of Quality Assurance and Software Testing; predominately the application of Quality Management Systems, BS 7925 and IEEE 829.
- It is the stated learning objective to provide preliminary preparation materials for the ISEB Foundation Certificate in Software Testing.
- It is the stated learning objective to provide further knowledge and skills competency covering those areas within Quality Assurance and Software Testing that are not detailed within the ISEB syllabus; in particular providing a specification for a comprehensive Quality Management System and providing an extensive selection of practical case studies and briefings in applied software testing, test planning and associated topics.

NB: ISO / IEC 29119 standard for Software Testing is due to be introduced from 2012 and combines both IEEE 829 and BS 7925; it is the intended replacement for these standards and also for IEEE 1008 Unit Testing. For more details see www.softwaretestingstandard.org

Eur Ing. Christopher Pugh, CEng. CSci. CITP. FBCS. FIAP.

Document intended for Public Distribution

No charge for download or usage

May 2012 - Version 16

Contents:

Pre-requisites and Assumed Knowledge

The content and nature of the QA training materials were intended to conform to the ISEB syllabus for both the Foundation Certificate and Practitioner Certificate in Software Testing, as well as, to comply with British Standards 7925 part one (Terminology) and part two (Component Testing) and IEEE 829 (Test Documentation Structure).

No formal pre-requisites and assumed knowledge is stipulated, however, BS 7925 and IEEE 829 should be referenced accordingly as they provide the foundations upon which the QA training materials were intended to build upon. It is strongly advised that these are printed and retained as part of a library of source literature.

The Information Systems Examinations Board or ISEB (www.bcs.org/iseb) is part of the British Computer Society which is the Chartered Professional Institution for Information Technology within the UK and is licensed by the Engineering Council UK and Science Council UK respectively.

ISEB certification leads to ISTQB certified testing qualifications. The Foundation Certificate in Software Testing now has dual accreditation with the ISTQB (International Software Testing Qualifications Board) and successful candidates will receive certification with both ISEB and ISTQB recognition at Foundation level. For more details of the ISEB in Software Testing see the BCS website (http://www.bcs.org/server.php?show=nav.6942).

Copies of the draft versions for BS 7925 parts one and two are available free from the Testing Standards website (http://www.testingstandards.co.uk/) which is sponsored by the BCS Special Interest Group in Software Testing (SIGiST).

Copies of IEEE 830 are available on-line from various sources, some are chargeable including www.softwaretestingstandard.org

NB: ISO / IEC 29119 standard for Software Testing is due to be introduced from 2012 and combines both IEEE 829 and BS 7925; it is the intended replacement for these standards and also for IEEE 1008 Unit Testing. For more details see www.softwaretestingstandard.org

The QA Management Model

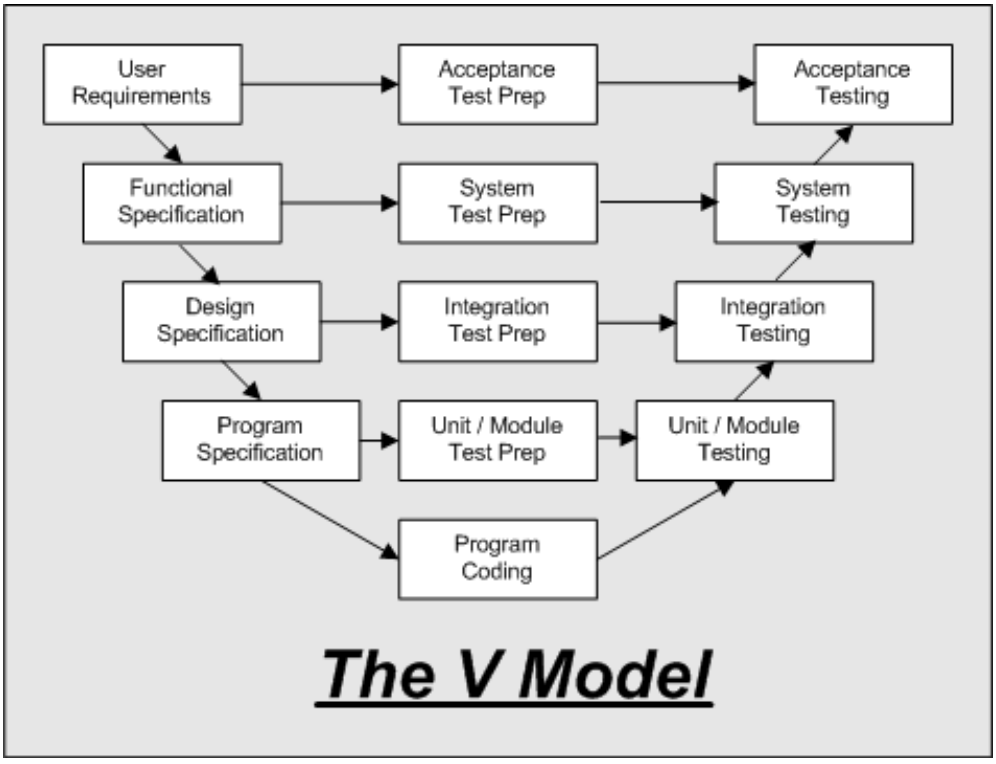
The QA Management Model was derived from the established V-model.

Essentially, the established V-model for Quality Assurance was historically introduced to address some of the serious problems associated with the more traditional Waterfall model of software development. Namely testing was conducted at the end of the lifecycle and often constrained by a lack of appropriate budgets, timescales, manpower, skillset and proper management.

Within the V-model, testing is not seen as a phase that happens at the end of development, instead it is recognised that for every stage of development an equal stage of testing is needed that often has its own set of documentation and procedures. Hence test preparation is not dependant on the code being delivered and so it can commence much earlier within the Systems Development Life Cycle (SDLC).

Other benefits of the V-model include …

- Testing is based on documentation that is completely independent of code.
- The appropriate tests can be created that can be validated and verified by others before actual use; thus encouraging knowledge transfer and communications between all parties.
- Errors are found much earlier and so the corrective costs are much cheaper, in fact more savings are made when errors are detected earlier rather than later.
- Quality and accountability is in-built into the product during its development and testing.

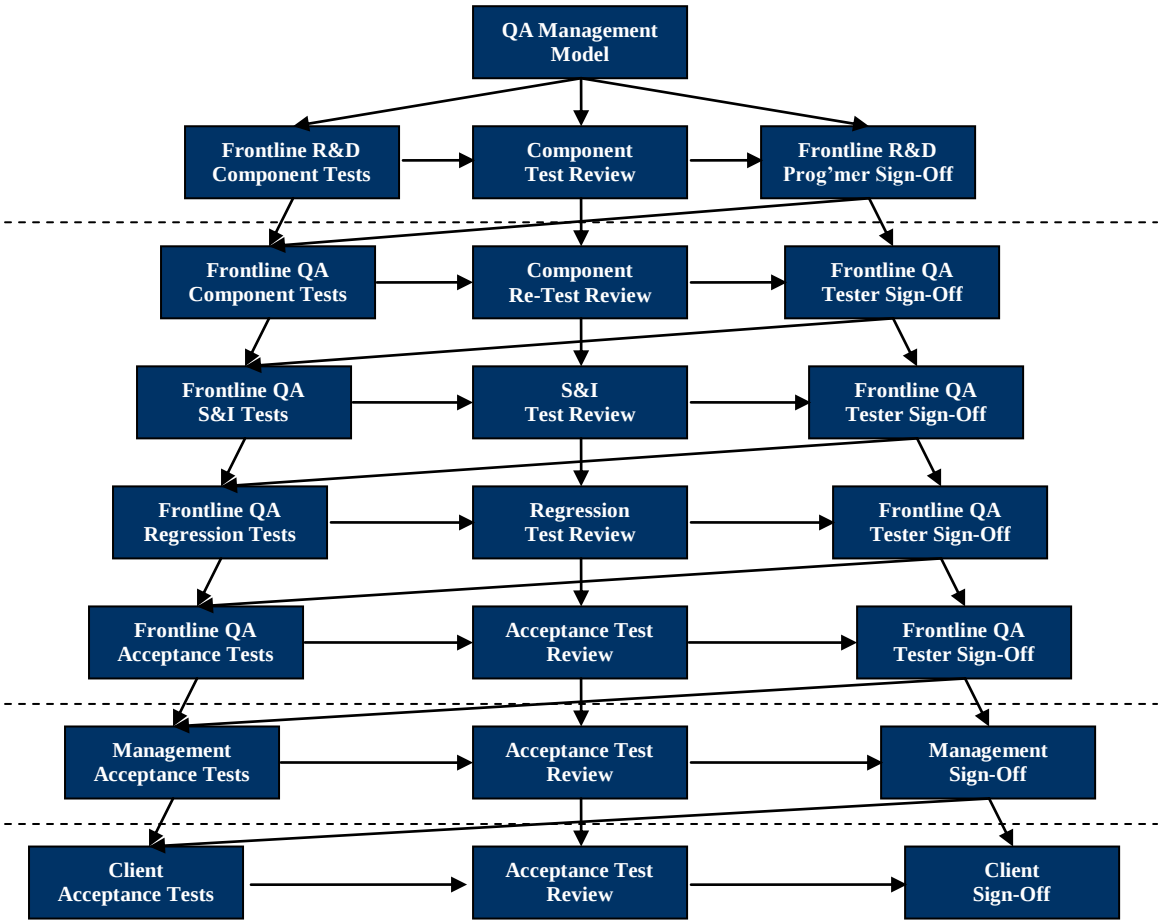


The implementation of the V-model is organisation specific and indeed it may require some rework before practical usage; particularly if being implemented as part of ISO 9001:2000.

An example of this being the QA Management model, which was developed from the V-model using inverse reciprocal transposition, where there are four basic levels; these being Frontline R&D, Frontline QA, Management and Client. Where an interface occurs between levels, as depicted by a dotted line, then the functions of the interface are replicated to ensure appropriate handover and communications. This is necessary to accommodate multiple languages, time-zones and geographic locations worldwide.

Essentially the focus is placed upon ensuring that front-line QA is effectively co-located and given suitable prominence at the point of front-line R&D delivery. Throughout testing is regularly repeated and expanded upon; incorporating individual component tests through to regression tests with acceptance tests being a subset of regression tests. Regression tests are constituted from both component tests and system & integration tests which have been combined together to form a collective script and rationalised accordingly to remove duplications, omissions and contradictions.

One example of a QA Management model, derived from the V-model, is depicted below …



Sample QMS design using the QA Management Model within the SDLC and ISO 9001

NB: The majority of the workload undertaken within the QA Management model is conducted by Front Line QA in regards Component Testing, Component Re-Testing, S&I Testing and Regression Testing. Front Line R&D is encouraged to provide the initial test scripts and technical documentation; which Front Line QA then duly refine and further define. Where these are not provided the Front Line QA then conducts the necessary QA Analysis to determine the test scripts from existing documentation such as User Manuals and the current system operation. The tests conducted by Management and Clients are normally undertaken using only a sample subset of the test scripts and are intended as inspectorate and sign-off mechanisms.

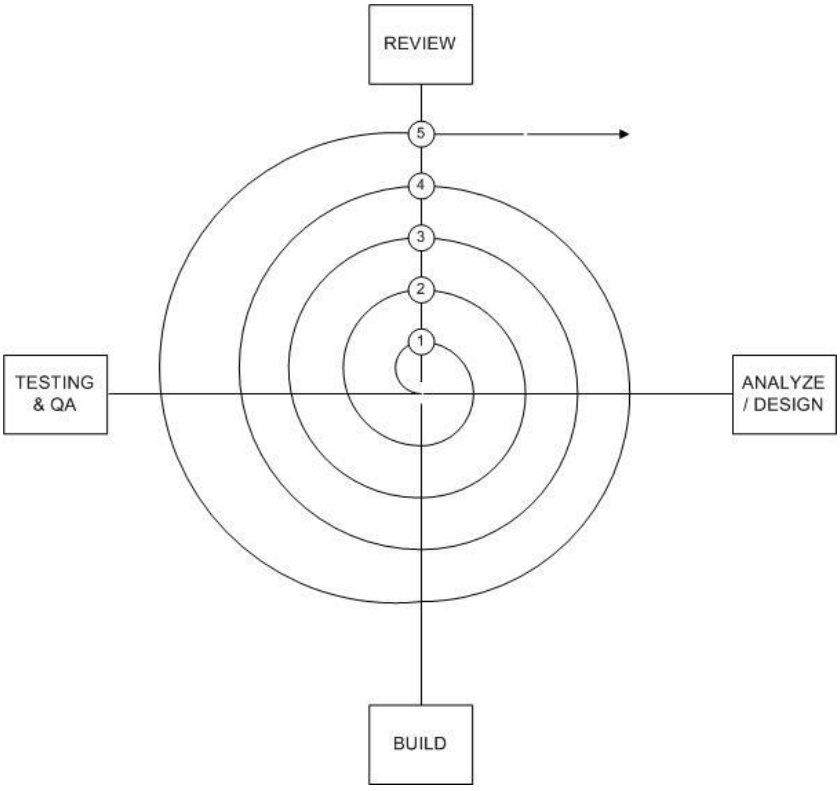
Terminology: R&D - Research and Development / QA - Quality Assurance / S&I - System & Integration

The practical implementing of the QA Management Model would be through a prescribed methodology consisting of several phases, each phase representing a specific stage of the Systems Development Life Cycle (also known as the SDLC).

A sample implementation of the phased Systems Development Life Cycle would be as follows ….

- Phase One: Planning / Initial Requirement Gathering.
- Phase Two: Functional Definition / Refining of Requirements.
- Phase Three: Design.
- Phase Four: Implementation.
- Phase Five: Component Test.
- Phase Six: Integration and System Test.
- Phase Seven: Alpha Test.
- Phase Eight: Beta Test.
- Phase Nine: Release.
- Phase Ten: Production.
- Phase Eleven: Ongoing Review.
- Phase Twelve: Retirement.

It is essential to point out that Software Development and Testing is normally implemented within an incremental and iterative lifecycle, where by several sets of phased deliveries are released following a standard Spiral model for system prototyping constituted by Analysis / Design, Build, Testing / QA and Review.



Indeed it is normally the case that the Design, Implementation and Testing phases are all combined together into a single Phase to create a series of incremental cycles. Thus beginning development with a basic software framework, then some functionality is added, and then more functionality is added, and so on until the final production version. However, for convenience the Design, Implementation and Testing phases will be considered as separate discrete phases with separate deliverables and entry / exit conditions.

Thus the correspondence between the V-model and the phased SDLC would be as follows …

Spiral Model	V-MODEL	Corresponding SDLC Phases
• Analyse/Design (Specifications)	• User Requirements.	• Planning / Initial Requirements Gathering.
• Analyse/Design (Specifications)	• Functional Specification.	• Functional Definition / Refining of Requirements.
• Analyse/Design (Specifications)	• Design Specification.	• Design.
• Analyse/Design (Specifications)	• Program Specification.	• Implementation.
Spiral Model	**V-MODEL**	**Corresponding SDLC Phases**
• Analyse/Design (Preparation)	• Acceptance Test Preparation.	• Planning / Initial Requirements Gathering.
• Analyse/Design (Preparation)	• System Test Preparation.	• Planning / Initial Requirements Gathering. • Functional Definition / Refining of Requirements.
• Analyse/Design (Preparation)	• Integration Test Preparation.	• Planning / Initial Requirements Gathering. • Functional Definition / Refining of Requirements. • Design.
• Analyse/Design (Preparation)	• Component Test Preparation.	• Planning / Initial Requirements Gathering. • Functional Definition / Refining of Requirements. • Design. • Implementation.
Spiral Model	**V-MODEL**	**Corresponding SDLC Phases**
• Build (Coding)	• Program Coding.	• Implementation.
Spiral Model	**V-MODEL**	**Corresponding SDLC Phases**
• Testing & QA	• Acceptance Testing.	• Alpha Test and Beta Test.
• Testing & QA	• System Testing.	• Integration / System Test.
• Testing & QA	• Integration Testing.	• Integration / System Test.
• Testing & QA	• Component Testing.	• Component Test.
Spiral Model	**V-MODEL**	**Corresponding SDLC Phases**
• Review		• Ongoing Review

Such an approach provides for the following beneficial outcomes …

- That the Test Preparation is conducted after the Systems Analysis & Design and before the commencement of the Coding.

- The Test Preparation is continually refined prior to commencing the Test Execution and any of the Test Preparation undertaken during earlier phases may be updated retrospectively (as required) so that full set of testing documentation is updated collectively. Hence the Acceptance Testing Preparation may be worked-upon whilst undertaking the Program Specification.

- That the execution order of the SDLC is linear whilst the V-model is hierarchal. Hence the Test Preparation phases can be undertaken at any time within the SDLC prior to the corresponding execution phase.

- It should be noted that the construction of Quality Management Systems (QMS) that are based on the V-model is openly documented and that other off-the-shelf QMS products could have been employed. For example **Watkins. J, 'Testing IT: An off-the-shelf Software Testing**

Process', Cambridge, 2001 is a very comprehensive QMS system based on the V-model that is publicly available and royalty free.

Thus by compartmentalising the Systems Development Lifecycle in to discrete phases then this permits the efficient and cost-effective management of the software development and testing. The underlying principle being that each of these phases needs to be completed, reviewed and approved before a project can move to the next phase.

Thus if we construct these phases into a formal definition, then:

- A phase is a subdivision of the product life cycle.
- At any point in the life cycle only one phase is formally active.
- Prior to the completion of the active phase the deliverables from the active phase are inspected and approved.
- A phase becomes active on formal completion of the previous phase.
- The set of deliverables from any given phase is fixed and defined by the Quality Management System.
- Any given project may omit or combine phases and / or deliverables and / or entry / exit criteria by prior arrangement with the Development Manager and QA Manager. However, such deviations must be documented within the project plan for appropriate communication and future reference.
- Work can commence on the phases ahead of the active phase, but they must be made retrospectively compliant with the results of the active phase and prior phases.
- A phase cannot be officially completed before the previous phase(s) have been completed in full.
- Throughout the necessary work is conducted in a proper and timely fashion that avoids incurring future 'development debt' by ensuring that the appropriate foundations for subsequent phases are laid within the current and prior phases.

This particular Quality Management System model comprises of twelve phases, and provides for a set of measurable events at each phase. The quality aspects are integrated throughout the project lifecycle, ensuring that product quality is built-in as the development progresses.

This division into phases not only helps to ensure the achievement of software product quality, but it is also intended for it to improve the quality of the business decision making processes and integration into ISO 9001:2000. By ensuring that the necessary information is provided throughout the lifecycle of a product and its associated product extensions, then the decisions made on the nature and cost of any particular development can be done so with the minimum amount of wastage in terms of time, effort and money. Furthermore the current and future phases of the development can be sustained from the defined deliverables from the earlier phases with clear entry and exit conditions that specify their scope and context.

Although this set of documentation and procedures provide for clearly defined and measurable Quality Assurance and Software Testing deliverables, the actual software lifecycle impinges at many points on other parts of the organisation and its operation. Other organisational processes (for example the preparation of marketing material and PR) would be conducted in parallel with the software development process and so the Product Manager and others need to co-ordinate the various concurrent threads of the project.

It should be borne in mind that a Quality Management System such as the one presented here is the ideal that is being strived for, however, due to the practicalities involved in Software Development and in particular within commercial environments, this ideal couldn't always be reached. Thus the implementation of a Quality Management System must be flexible and may be deviated from as necessary when the appropriate authority has been given (e.g. from the IT Director) and any deviation must be documented in the relevant information repositories for future reference (e.g. Project Plan).

Please Note:

- Due to the structure and discipline imposed by the Quality Management System, the effect will be to slow down the progress of the project to a steady and consistent pace. In addition some levels of bureaucracy and hierarchy will be necessary in order to maintain the project. This might be quite a culture shock to some people who are used to writing 'quick and dirty' code and from past experience it is likely that they will resist any changes that are imposed without full consultation, explanation and encouragement.

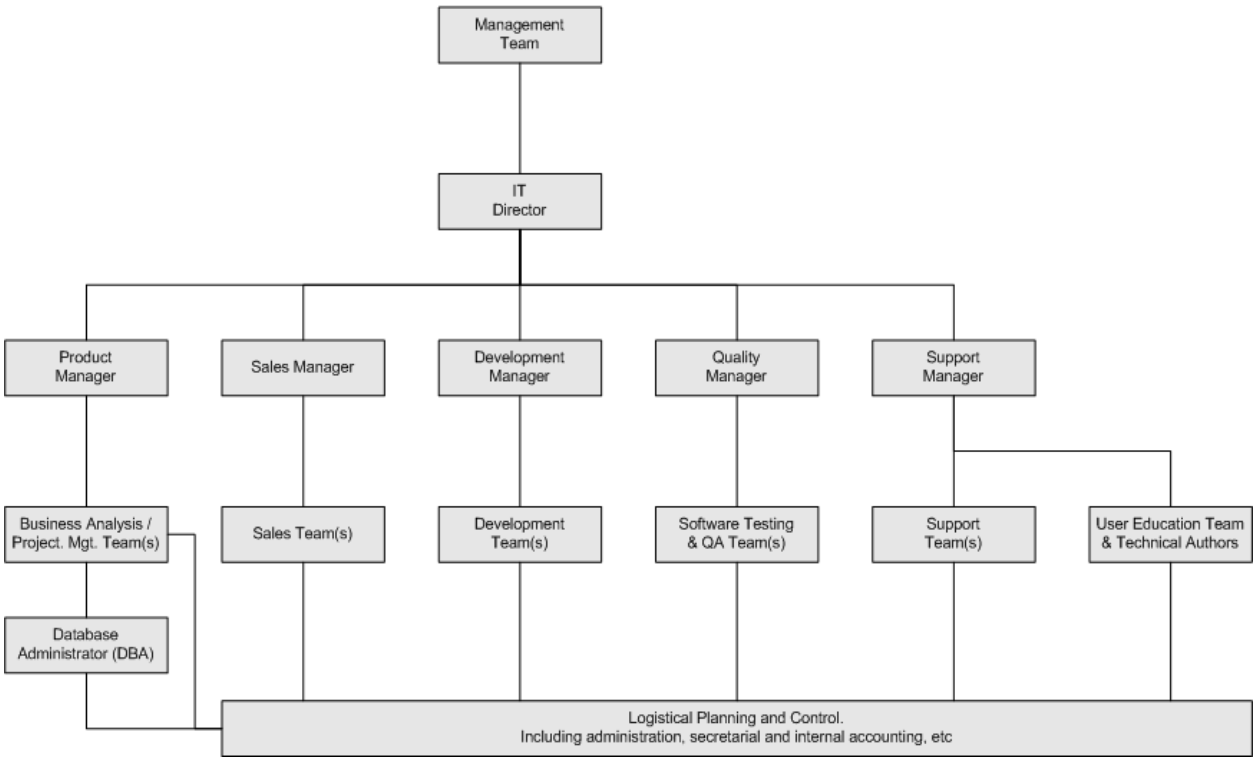
- Although it is permitted to deviate from the Quality Management System, it is still important that the procedures and documentation are not implemented in half-measures and should be undertaken from the project onset. In other words it is literally a case of all or nothing when implementing the Quality Management System. If the project is too small (i.e. a simple application), or drastic software development activities are being undertaken (e.g. as a result of emergency software re-hash), then it is better not to implement the Quality Management System at all, or at least wait until the project is in a stable state so that the additional workload in implementing the Quality Management System does not overload the staff or de-rail the project. It has been known for IT departments and whole organisations to be brought to a grinding halt in their operations while trying to implement badly conceived Quality Management Systems.

- Quality Management Systems are only useful if they are properly designed for practical use on a daily basis from the onset of the project. The Quality Management System specified here was based on 'best practices' and 'industry standards' such as the V-model and phased SDLC.

- As would be expected, the Quality Management System can only be successful if implemented as a team effort and the various key-players of that team are listed within the following section. A person may act in more than one role as deemed necessary; however, these should be documented and communicated to all relevant parties.

Key Players / Organisational Hierarchy

Term	Meaning
Management Team.	IT Director and 1st line reports (i.e. company executives and directors).
IT Director.	Overall head of software development.
Product Manager.	The person responsible for a product or product line and who is tasked with business planning, liaising with sales and customers, as well as, determining what new features or functionality is to be added to future versions. The Product Manager also performs any Project Management duties as deemed necessary for the product development and may be assisted by a team of Project Managers and Business Analysts.
Development Manager.	The head of the software development department.
Development Team(s).	A team of people working on the software design and development for any given product line.
QA Manger.	The person responsible for Software Testing and Quality Assurance (QA). In particular responsible for auditing all deliverables and ensuing that each phase is completed accordingly.
Software Testing / QA Team(s).	A team of people working on the software testing and QA for any given product line.
Support Manager.	The person responsible for support and order fulfilment.
Support Team(s).	A team of people working on the software support and order fulfilment for any given product line.
Logistical Planning & Control.	Administration, secretarial and internal accounting, etc.
User Education Team & Technical Authors.	A team of technical authors and software trainers. Input is also provided by other stake-holders within the project.
Database Administrator (DBA).	The person responsible for database design and maintenance within the organisation.



The Twelve Phases within the SDLC

The following components provide the documentation and procedures that define the Quality Management System (QMS) as prescribed.

- Phase One: Planning / Initial Requirement Gathering.
- Phase Two: Functional Definition / Refining of Requirements.
- Phase Three: Design.
- Phase Four: Implementation.
- Phase Five: Component Test.
- Phase Six: Integration and System Test.
- Phase Seven: Alpha Test.
- Phase Eight: Beta Test.
- Phase Nine: Release.
- Phase Ten: Production.
- Phase Eleven: Ongoing Review.
- Phase Twelve: Retirement.
- QA Documentation.

The format for defining each Phase of the Systems Development Life Cycle (SDLC) will be through a summary table specifying the Phase Entry Conditions, Phase Exit Conditions and Phase Deliverables. Each summary table will be accompanied by associated narrative items that provide a more detailed description. Only the critical items are specified for the Phase Deliverables as it assumed that the documentation from the previous phases will provide the inputs for the current phase and that all documents will duly be updated and maintained collectively, however, it is only the critical items that the QA Manger has to audit and these should be reviewed and approved as part of the phase activities.

Approving Authority

Concerning the approval authority within the QMS system, the minimum approval authority is specified below for minor and major changes; however, it is assumed that higher authorities within the management structure of the organisation will have precedence. The following table provides a summary of the minimum authority for each Phase within the SDLC employed by this particular QMS system.

APPROVING AUTHORITY	SDLC Phase											
	01	02	03	04	05	06	07	08	09	10	11	12
Management Team.												
IT Director.	(+)	(+)	(+)	(+)	(+)	(+)	(+)	(+)	(+)	(+)	(+)	(+)
Product Manager.	(-)	(-)	(+)	(+)	(+)	(+)	(+)	(+)	(-)			
Development Manager.	(-)	(-)	(-)	(-)	(-)	(-)	(-)	(-)				
QA Manger.					(-)	(-)	(-)					
Support Manager.										(-)		

Legend:

(+) Approval required for major changes / new features.
(-) Approval required for minor changes / new features.

Phase One: Planning / Initial Requirements Gathering

SDLC Phase One:	• Planning and Initial Requirements.
Phase Entry Conditions:	• There is no fixed protocol for dealing with entry conditions of the Planning Phase as there may be many triggers. For example an executive decision by the Management Team for new features / functions or an approval by the Development / Product Manager for the instigation of bug-fixes and enhancements, etc.
Phase Exit Conditions:	• An executive decision has been made approving or terminating the continuation of the project, based on the viability of the Development costing, the proposed product features and the business justification as stated within the Draft Business Plan / Project Plan. • The QA Manager has audited the phase deliverables.
Phase Deliverables: (Critical Items Only)	• Draft Business Plan / Project Plan. • Draft User Requirements. • Draft Functional Specification.

- The Product Manager works with the Development Team and Development Manager to define the overall product parameters and end-user / client requirements, as well as, stipulates who will be held responsible during the various phases of the systems development lifecycle for the project. Please note that it has been assumed that within this SDLC that the Development Team is also responsible for the Business & Systems Analysis / Design function. This may not always be the case and some organisations would have their own separate teams of Business Analysts and Project Managers who would undertake the Business & Systems Analysis / Design function and would also separately manage the various Business Plans and Project Plans.

- A package of information is assembled that defines the top-level functionality, the outline business case and an estimate of the project costs and manpower requirements.

- During the Planning Phase, it is the stated aim to spend only the minimal amount of the product's development time and cost in confirming if it is worthwhile to proceed with the project from the business perspective.

- During the Initial Requirements Gathering, all the requirements known at the time are collected, reviewed, prioritised and stored in a convenient repository for future retrieval and analysis.

- The Planning Phase should trap projects whose cost clearly cannot be justified by the predicted returns (i.e. cost savings, ROI and functional benefits / features, etc). More finely balanced business cases will probably need to go to the second phase (i.e. Functional Definition / Refining of Requirements) and may possibly include a proof-of-concept version to be built or an existing system / sub-system employed to act as an actual working prototype.

- Rejected project proposals are still retained for information and historical reference purposes; indeed they may even be resurrected at a future point after some rework or they may provide the necessary preliminary work for other projects.

Phase Two: Functional Definition / Refining of Requirements

SDLC Phase Two:	• Functional Definition and Refining of Requirements.
Phase Entry Conditions:	• Completion of the Planning Phase.
Phase Exit Conditions:	• The Functional Specification is reviewed and approved. • The Project Plan is reviewed and approved. • User Requirements are reviewed and approved. • The IT Director agrees any changes to the resourcing and scheduling of the project. • The QA Manager has audited the phase deliverables.
Phase Deliverables: (Critical Items Only)	• Revised Business Plan / Project Plan. • Revised User Requirements. • Revised Functional Specification.

- Using the Draft Functional Specification defined in the previous phase a Functional Specification of the product is produced. This defines the operation of the product from the user's viewpoint. It enables the programmers to begin work on the design of the implementation (and in addition the technical authors to begin work on the user documentation).

- During the Functional Definition phase the Requirements are further defined and refined and the Functional Specification modified as necessary. In particular Quality Assurance and Software Testing criteria are added, including 'Success / Exit' criteria for each requirement so that their implementation can be evaluated accordingly.

- Although the Functional Specification is normally a document, it can take any appropriate form. This includes GUI interfaces, which in particular may be more efficiently specified with a design tool (e.g. Ms Visio, etc), rather than drawing them on paper. Equally appropriate is the trial implementation of a proof-of-concept prototype or an interim version of the product. Thus the essential process is to capture and evaluate the required functionality of the intended product. Hence it is known exactly what is going to be implemented in the new system or sub-system, and it also provides an effective method of communicating this information to the rest of the organisation and to external parties. Also favoured are existing (or third-party) products that can be evaluated and then reverse-engineered back into their schematic designs and original concepts for inclusion within the new product.

- During this phase the Product Manager also updates the Project Plan with the progress of the project. The Project Plan details the tasks to be performed, the resources to be allocated and the projected costs and time scales involved. It also notes any external dependencies the project may have, as well as, any risks and critical success factors. The Project Plan defines any changes to the standard life cycle model that may be required.

- The IT Director, Project Manager and Development Manager should review the User Requirements, Functional Specification and Project Plan and provide any input as necessary. Others may be invited to meetings including the Developers, the QA Manager, the Sales Manager, the Support Manager and Client representatives as deemed appropriate. Any major deviation from the required functionality or cost must be approved by the IT Director, whereas minor deviations may be approved by the Development Manager or Product Manager as appropriate.

- The Product Manager is now fully equipped to communicate both the development cost and the functionality of the product. This information is the trigger to a number of important business processes ranging from sales, to strategic marketing of selected customer groups and corporate budget planning.

It cannot be stressed enough the benefits of working from a documented functional specification that has been mutually agreed upon with the Client. Although, as with any agreement, some flexibility is required, however, it should always be borne in mind that regardless of how valuable a specification, it still provides no guarantees of success.

'Most currently recommended methods for defining business system requirements are designed to establish a final, complete, consistent, and correct set of requirements before the system is designed, constructed, seen or experienced by the user. Common and recurring industry experience indicates that despite the use of rigorous techniques, in many cases users still reject applications as neither correct nor complete upon completion. Consequently, expensive, time consuming, and divisive rework is required to harmonise the original specification with the definitive test of actual operational needs. In the worst cases, rather than retrofit the delivered system, it is abandoned. Developers may build and test against specifications but users accept or reject against current and actual operational realities.'

Boar, B,. 'Application Prototyping', Wiely-Interscience, 1984

NB: At this conjecture the Business Plan is no longer part of the Project Plan and it is continued separately outside of the Systems Development Life Cycle. The Business Plan was originally instigated to provide justification for commencement of the project and therefore should still be maintained on a concurrent basis in order to provide justification for continuation of the project and for detailed analysis of the costs, benefits and resources involved. In particular the costs and schedules may be planned and maintained using graphical computer software such as Microsoft Project employing PERT (Project Evaluation and Review Technique).

HINT: Often it is extremely difficult and costly in terms of time, money and manpower to document the existing system by reverse engineering its design back into a formal Requirements Specifications and Functional Design. However as a viable alternative the existing 'working' version of the software may be presented as the specification including all associated user / technical documentation, QA / test scripts, source code and working install versions, etc. Also appropriate would be printed copies of all components within the user interface; this includes screens, menus, reports and pop-up's, etc. Likewise these would also be supplemented by 'prototype' versions of any new functionality with a full write-up explaining in detail its intended operation.

Phase Three: Design

SDLC Phase Three:	• Design.
Phase Entry Conditions:	• Completion of the Functional Definition Phase.
Phase Exit Conditions:	• The Design Specifications have been reviewed and approved. • The Project Plan has been reviewed and approved. • The Test Plan has been reviewed and approved. • The QA Manager has audited the phase deliverables.
Phase Deliverables: (Critical Items Only)	• Design Specifications. • Test Plans. • Project Plans. • Initial User Documentation and Help Text Synopses.

- It is not the purpose of this publication to define how the Design of the system is specified and documented, instead its intended purpose is only to specify the QA aspects in terms of documentation and procedures. Suffice to say that the key aim of this phase is to produce a documented product design to a level of detail that is correct, coherent and capable of being implemented to meet the functionality described in the Functional Specification. The design will be used both in the implementation phase and to facilitate later testing, maintenance and enhancement to the product.

- As already referred to within the Functional Definition / Refining of Requirements, the method that the product design is captured can vary considerably from project to project. The Project Plan should state exactly which methodologies and techniques are to be used, and what form the resultant design will take.

- The Design Phase includes the compilation of the necessary Test Plan which underpins the quality goals that are intended for the project. The Test Plan describes the sequence of test milestones and techniques that will be employed to verify and validate the project progress and quality at various key points through its development life cycle.

Phase Four: Implementation

SDLC Phase Four:	• Implementation.
Phase Entry Conditions:	• Completion of the Design Phase.
Phase Exit Conditions:	• All Phase Deliverables provided by the implementation activities are reviewed and approved. • The QA Manager has audited the phase deliverables.
Phase Deliverables: (Critical Items Only)	• The Program Specification. • The Product Source Code. • Product Test Preparation. • System Build & Object Code Compilation. • User Documentation and Help Text. • Install Mechanism.

- All program code required by the product design and its installation software are produced during this phase. Work also continues on the user documentation and help text.

- It is important that all aspects of the project implementation are kept in perspective and not swept ahead to be done at a later date, when there is (apparently) more time (however, in reality there rarely usually is). In essence avoid stocking up on future development debt. A successful product test will only follow from having a completed product to perform the necessary tests on. Missing components cannot be tested and so their testing will have to be scheduled when they are complete and available. Indeed untested components may severely impact or even jeopardise a project and so they should be properly managed; ideally their priority should be escalated so that they can be tested thoroughly at the earliest available opportunity.

- Any technicalities and practicalities to overcome should have been identified within the preliminary phases and duly addressed before reaching the implementation phase. Failure to do so can cause expensive delays, cost overruns and project derailments.

- A review of the Test Plan is undertaken at this stage to ensure all previous test preparation is complete and updated retrospectively as a collective prior to commencement of the actual testing activities.

- It is expected that the programmers in R&D will perform Component Testing on their own code before it is released for Component Testing by QA; including the provision of associated test scripts and technical documentation employed. This considerably helps to prevent sloppy ‘quick & dirty’ coding from being produced. Ideally the software is to be demonstrated as working by the Software R&D team in the presence of the Software QA team representatives BEFORE the package of work is accepted for progression on the next phase! All too often a code build is delivered late by Software R&D and in a work-in-progress state; in other words somewhat ‘incomplete’ and ‘buggy’.

- Depending upon the organisation there may be several programming streams in operation, however, before compilation of a build prior to conducting Component Testing, the various streams should be brought together to form a single definitive version of the source code. Hence all the source code for the product is to be ‘checked-in’ into a common repository for source code control such as Ms SourceSafe, CVS Eclipse or StarTeam4; this ensures that the program code is complete and also that it is the latest version. This provides a common base for re-distribution as necessary after completing its Integration and System Tests.

Phase Five: Component Test

SDLC Phase Five:	• Component Testing.
Phase Entry Conditions:	• Completion of the Implementation Phase.
Phase Exit Conditions:	• The requirements of the Test Plan are met for the Component Testing. • The QA Manager has audited the phase deliverables.
Phase Deliverables: (Critical Items Only)	• Component Test Reports.

- The aim of the Component Test is to demonstrate the correct operation of the product (or component of the product) that has been produced during the Implementation Phase. The Component Test is also known as a Module or Unit Test and is extensively documented within industry standard BS7925 part two.

- The Test Plan should have already defined the tests that will be run, the data to be used, the expected results and the method of reporting the results. This would have been undertaken within the test preparation which should have been reviewed as part of the Implementation phase. An industry standard for the Test Plan being the IEEE 829. The aforementioned is the ideal; however, often this is not possible but the testing and its planning should always be conducted in timely and proper fashion encompassing all phases of the Systems Development Lifecycle (SDLC).

- A database of known problems should be created / updated (e.g. on a call logging / support system) and then maintained for the remainder of the life cycle from which Software QA Metrics can be extracted.

- The Component Test is one of the most critical points in the life cycle. Great care and attention to detail has to be taken in order to ensure that the product (or component of the product) is well specified to an appropriate level and that the testing provides formal validation and verification that it has been correctly implemented. Any errors or incompleteness of the product (compared to its specification) are rectified at this point or if significant changes are necessary then Phase Four is revisited.

- By agreement in the Project Plan, it is permissible to allow incremental and iterative software development to take place as part of a prototyping approach, such as the Spiral model. This is achieved by combining the Design, Implementation and Testing phases. Thus the initial software framework is developed and then the functionality is added in repeated steps until the product (or a component of the product) is complete and is ready for official Product Testing.

- If the Component Test conducted by the programmers appears to have worked without issue and yet the build failed its Component Test by the QA team, then this is indicative of the source code diverging between different programmers (i.e. normally as a result of branching or different development streams). That is to say that the programmers will often be working independently on separate projects and so the impact of code changes or additions made by others would not have been considered in their testing. Likewise it is often the case that programmers will make local changes to their code and not consider the impact with their or others coding. In these circumstances it is necessary for the programmers to redo their Component Testing, however, using synced set of code. That is to say a merged set of source code which has been brought together to form a single definitive version for that build; after which the programmers may resume their Component Testing prior to hand-over to the QA team. Other reasons for a difference in the results of the Component Testing between programmers and the QA team being system configuration and initial database state. Again this can be easily addressed through proper documentation and direction from the programmers as to the necessary preparation of the test environment.

Phase Six: Integration and System Tests

SDLC Phase Six:	• Integration and System Testing.
Phase Entry Conditions:	• Completion of Component Test Phase.
Phase Exit Conditions:	• The requirements of the Test Plan are met for Integration and System testing. • The QA Manager has audited the phase deliverables.
Phase Deliverables: (Critical Items Only)	• Integration and System Test Reports. • Draft User Manuals and Help Text.

- The preceding phase validated and verified the operation of a single component. Integration and System Testing proves that all of the components of the product work correctly both individually and collectively as a complete package throughout the life-cycle of the application or its component sub-set. In other words, in the case of an Invoicing system the invoice creation, subsequent modification, printing, posting to Sales Ledger and archiving would all be tested, even if it was only the invoice amend screen that was modified and required retesting. Hence the level of testing should not just be targeted at the system component or sub-component that was developed, modified or enhanced; instead the operation of the product should be also tested within its larger environment so that any repercussions or side-effects can be identified. For smaller projects the Component Test and the System Test may overlap to such an extent that the Test Plan covers as a single phase.

- This is also the time when the Development Teams get their first chance to review the operation of the product against the original requirements, as well as, enabling them to get a first indication of operational parameters such as actual performance and usability.

- User manuals and help text should have reached the full and complete draft at this stage and should be checked against the product during the system testing.

- A database of known problems should be created / updated (e.g. on a call logging / support system) and then maintained for the remainder of the life cycle. During the Integration and System Test, if any residual faults are identified then they are removed, however, if any major errors or incompleteness of the product (compared to its specification) are found or if significant changes are still necessary then Phase Four through to Phase Five are revisited.

- At the end of the System and Integration Test the development of the product is complete. The rest of life cycle is concerned with checking and improving the quality of the result.

- The source base may now be branched as required. Any further bug fixes arising from subsequent testing will be built into the branched source and resubmitted at the Component Test level. This allows development of new features to continue whilst retaining the ability to react quickly to problems found in the field. Please note that not every organisation may wish to branch their source code at this point due to the complexities it can bring to the version control mechanism.

Phase Seven: Alpha Test

SDLC Phase Seven:	• Alpha Test.
Phase Entry Conditions:	• Completion of Integration & System Test Phase. • All product deliverables are available.
Phase Exit Conditions:	• The requirements of the Test Plan are met for Alpha Testing. • The QA Manager has audited the phase deliverables.
Phase Deliverables: (Critical Items Only)	• Alpha Test Report. • Beta Software with User Manual & Help Text. • Installation Software.

- Alpha Testing is defined by BS7925 part one as being a simulated or actual operational testing at an in-house site not otherwise involved with the software developers.

- Before 'Live' trials are conducted, the software must be thoroughly tested in-house for a suitable period and those who issue its release must be satisfied with its operation. End users will always find program 'issues' either through misuse of the system or they will spot logic mistakes and misinterpretations by the programming team, however, through proper in-house testing the severity and occurrences of these 'issues' can be minimised. Within Phase Eight the BETA trials of the software at selected sites are conducted within 'live' environments; the participating clients and their end-users do not take lightly to disruptions or computer system failures.

- Regression testing is the ideal approach for the Alpha Test. That is to say the Regression Test scripts are constituted from both Component Test and System & Integration scripts which have combined together to form a collective script and rationalised accordingly to remove duplications, omissions and contradictions. The Regression Test scripts are executed in accordance with the Test Plan and are intended to exercise the system in both breadth and depth.

- Also at this stage the product is ready to be exposed to staff outside of the test team (i.e. usually the Support, Training and Sales staff), albeit in a controlled environment. The aim is to add randomness to the test regime that is very hard to get from bench tests. It is also an opportunity to get some early user reaction to the usability of the product. Remembering that by nature programmers can often make the worst software testers for their own products, as it is usually the case that they will subconsciously operate the software as it was originally intended to be used; and not give it the general abuse and misuse that an actual end-user would.

- At the onset of the Alpha Test it is necessary that the Support, Training and Sales staff be trained in the full use of the product otherwise they cannot be expected to conduct their task properly. This training process has a number of useful repercussions, not only are staff trained but many incidental bugs and faults have an embarrassing habit of appearing during the training session that had previously eluded detection in earlier phases of testing.

- Those involved in the Alpha Test commits to providing reports of any faults that they encounter, as well as, providing information on their use of the product. Any problems that still impedes testing will have to be fixed; if major work is required then the Phase Four through to Phase Six are revisited. The database of known problems is duly updated (e.g. on a call logging / support system).

- Prior to commencing the Alpha Test phase it is necessary for the complete ‘package’ to be assembled including the product software, user documentation and automated installation software. Included within this list would be additional items such as CD Labels and Covers (to be designed and printed) along with any necessary installation instructions and covering letter. Likewise with software distribution websites and downloadable update services. All of which must receive the full approval of the Product Manager before release.

- The Alpha Test Programme is co-ordinated by the QA Manager and who in turn liaises with the Product Manager and the Development Manager. The Test Plan will determine how the Alpha Test is managed and what the conditions are specified for a successful phase exit.

Phase Eight: Beta Test

SDLC Phase Eight:	• BETA Test.
Phase Entry Conditions:	• The product meets the requirements of the Test Plan for successful completion of the Alpha Test. • IT Director approves the start of the Beta Test phase with distribution to selected 'live' sites.
Phase Exit Conditions:	• The requirements of the Test Plan for the Beta Test are met. • The QA Manager has audited the phase deliverables.
Phase Deliverables: (Critical Items Only)	• Beta Test Report.

- Beta Testing is defined by BS7925 part one as being operational testing at a site not otherwise involved with the software developers. Typical Beta Testing is undertaken by selected Customers who have agreed to operate the new versions of the program within their 'live' environment.

- The aim of the Beta Test phase is to deliver the completed product to a select set of customers (i.e. trial sites – also known as Beta Sites), simultaneously exercising both the product and the company processes. Technically speaking the aims are very similar to the Alpha Test, however, the product is being subjected to real live use by the Clients and their End-Users, during which feedback about both product errors and product usability are actively sought from the 'trial sites'.

- The aim of a beta test is not to find program errors (as they should have been identified and resolved before this stage), it is in fact to just confirm that there are NO known errors and to gauge the reaction of the clients. No unstable or incomplete product should go to beta testing, as otherwise there would not be any relevant benefits or feedback; instead it would only seriously annoy the trial sites and may even result in the loss of their custom. Hence the authority of the IT Director is required before its release as the IT Director has ultimate responsibility for the product.

- At this stage in proceedings, the sales, support and distribution functions should be involved. The Beta Test phase also importantly checks that the organisation's supply and support processes are in place before the product goes on to general release.

- During the Beta Test, the product is distributed and supported as if it were fully released. Only the extent of the product distribution is controlled; by limiting access to customers (i.e. trial sites) who can make valuable contribution to checking product quality.

- During the Beta Test, if any residual faults are identified then they are removed, however, if any major errors or incompleteness of the product (compared to its specification) are found or if significant changes are still necessary then Phase Four through to Phase Seven are revisited. The database of known problems is duly updated (e.g. on a call logging / support system).

- The Beta Test Programme is co-ordinated by the QA Manager and who in turn liaises with the Product Manager and the Development Manager. The Test Plan will determine how the Beta Test is managed and what the conditions are specified for a successful phase exit. Throughout the IT Director is kept fully informed as to the progress as it is the IT Director who authorises its live release to Beta sites.

Phase Nine: Release

SDLC Phase Nine:	• Release.
Phase Exit Conditions:	• Product Masters are available. • The IT Director agrees that the product has successfully completed the Release Phase. • All relevant Support, Sales and Product Distribution staff are fully trained on the product. • The QA Manager has audited the phase deliverables.
Phase Deliverables: (Critical Items Only)	• Product Masters of all software and documentation. • Final Test Report. • Release Notes.

- This phase produces the first batch for the product release.

- Once all the support, training, sales and product distribution departments are fully brought up to speed, then the Sales and Training Departments can actively promote the product to the customer base without restriction.

- All documentation is reviewed and collectively updated; particularly the Business Plans / Project Plans, User Requirements, Functional Specifications, Design Specifications, Program Specifications and Test Plans. Likewise the Technical Manuals and User Manuals should reflect system changes and new additions.

Phase Ten: Production & Client Implementation

SDLC Phase Ten:	• Production.
Phase Exit Conditions:	• Product is due to be retired.
Phase Deliverables: (Critical Items Only)	• Product Masters. • Completed User Documentation & Help Text. • Release Notes.

- This phase represents the continuing life of the product. Most software will require regular updates to include bug-fixes, enhancements and general maintenance. Depending on the circumstances and severity of the work required then Phase Four through to Phase Nine are revisited.

- It is important that updates relating to bug fixes are not rushed out for every problem that is encountered. Instead, identify 'work-around solutions' or 'avoidance procedures'. These can be easily issued to the users so that they can include them within their manuals. If there are only a few customers then issuing updates is viable, however, it can seriously disrupt and annoy the end-users. When there are more than several customers, this option becomes a logistical nightmare and so it is best to wait until the next official release.

- Furthermore, never forget that the end-users are operating 'live' systems. It is important to place one's self in their shoes. The biggest fear they have is that the system will lose or corrupt their data and also disrupt their daily operations. This in turn could ultimately cost them their business; hence, their concerns and queries must be handled with due respect and attention, as well as, temperance, if the exchanges become heated. This situation is particularly relevant with software upgrades and system patches as the end-users can become alarmed if they receive new versions of the program on a frequent basis. It has been known for some end-users to misunderstand the situation and believe the previous versions must be faulty and, as a result, they assume that some of their data must therefore be potentially missing or corrupt. Calming the customer down in these cases is not an easy task and so this kind of situation should be avoided by regular, but in-frequent updates with a full explanation of the benefits gained from the changes to the system.

- Caveat Emptor … Buyer Beware! Concerning client site implementation, this is a very grey area between the software supplier and their clients that can cause friction and problems between the two parties. Ideally it is necessary to encourage clients to implement their own appropriate QA policies and to take responsibility for the systems they use, particularly if the software products are mission critical to the business. Regardless of whether the application is being supplied ‘off-the-shelf’ or ‘bespoke’, it is necessary that the clients and their end-users have confidence in the product and know that it works correctly within normal operating parameters. The clients should be encouraged to establish their own team of ‘expert users’ who provide front-line product support, training and testing. Ideally a single person of authority (i.e. a ‘champion’ / ‘facilitator’) and their deputy should be designated at both the supplier and client ends, through which all day-to-day communiqués and agreements are conducted. This is a proven approach to managing manpower resources for both the client sites and the software provider, particularly as it allows for issues to be dealt with very quickly and efficiently. In addition, a system of phased implementations should be encouraged as this provides the client sites with the opportunity to try-out the software products in a controlled ‘test’ environment before actual ‘live’ use. For a staggered separation between phases, also known as sandboxing, it is essential that each distinct phase should have its own individual copy of the database that is independent of the other databases and ideally encapsulated within a virtual environment.

  Phase1 Product install disks, documentation and release / testing procedures are assembled ready for distribution by the Software Supplier. In particular the system configuration / parameter data and appropriate ‘fixes’ must all be available in full and ready for use. Ideally the release / testing procedures should also be issued by the Software Supplier, along with training and guidance notes to assist the Client in their UAT. It has been assumed that the Software Supplier has already implemented the release on their test systems and has certified that the release is ready for distribution to the Client sites.

Phase 2 Using the technical documentation from the Software Supplier as a source of reference (i.e. the release / testing procedures), the 'expert users' at the Client site checks that the new / updated software installation procedures can be successfully installed and operated on the computer systems at their site.

Phase 3 The 'expert users' at the Client site undertake their own comprehensive testing of the new / updated software to ensure it meets their minimum quality standards. Ideally they should try out all the various functions of the system. Every menu option, screen-view and printed-report should be run to ensure that the results make sense and that data is being saved & recalled as expected. It is also necessary that the management at the client sites review the documentation and testing undertaken by their 'expert users' and also to review the operation of the new / updated software in terms of its business perspective and impact.

Phase 4 Once the new / updated software has been fully verified and validated it can then be implemented by the Client Site. At this point the 'live' implementation should be a smooth transition. The 'expert users' at the client sites then provide the necessary product training and support to the other end users at their specific site.

Phase 5 It is a requirement that the Client Site informs the Software Supplier of the successful implementation of the release version and confirms their Acceptance as part of a UAT protocol. Likewise they should report any issues or concerns they experience during the installation and subsequent usage; these will be recorded on a database of known problems (e.g. on a call logging / support system) and acted upon accordingly.

Phase Eleven: Ongoing Review

SDLC Phase Eleven:	• Ongoing Review.
Phase Exit Conditions:	• Not Applicable.
Phase Deliverables: (Critical Items Only)	• Metrics. • Task Group Reports.

- Throughout all of the phases of the life cycle model, an ongoing review is continually undertaken by a specially appointed stake-holder 'task group' headed by the IT Director to evaluate project progress through the implementation of Metrics.

- The stake-holder 'task' group (also known as a 'steering' group) is constituted from a representative sub-set of the client-base along with the representatives from the development team, support team, training team, sales/marketing team and management team. The planned remit is to drive the ongoing development from initial prototype through to the live product using an incremental and iterative cycle of refinement and further definement. The input for the 'steering' group will be the current working prototype / version and associated documentation, formal system requirements, client / end-user feed-back and the periodic evaluation of previous versions / competitor products. Throughout the necessary Task Group Reports will be compiled and distributed accordingly.

- The use of Software Quality Metrics is far from new, in fact it is highly recommended. Software Quality Metrics are most useful when they have been base-lined at regular intervals and the statistical trends analysed, however, it may take several weeks of data sampling before the results are accurate and reliable. Although, once this initial 'tooling-up' period is complete, all the effort and expense invested does soon repay itself many times over by highlighting the various 'weak areas' within the software development and testing, as well as providing official statistics that can be used to attest to others the quality of the software. For example, Section 4 of BS7925 part two contains extensive details of various Software Quality Metrics that can be employed within a micro environment (i.e. at the programming level), however, such metrics can be difficult to compile and are prone to people's differing interpretations and personal agendas. Often a more appropriate approach is to analyse the Software Quality Metrics for logged issues such as 'bugs' and 'new features' which can then be later rationalised for public distribution, particularly for project management, marketing and PR purposes. Software Quality Metrics can be presented to clients and the end-users alike in order to gain both their trust and support in the delivery of new or upgraded products. In doing so good relations and product confidence can be steadily built upon solid foundations that will steadily grow if appropriately managed. Likewise valuable cost / time / manpower statistics can also be extracted that provides a reliable benchmark when compiling estimates for future testing activities. For more details see Briefing on Software Quality Metrics.

Phase Twelve: Retirement

SDLC Phase Twelve:	• Retirement.
Phase Exit Conditions:	• A product cannot exit this phase. Instead it can only be re-released (i.e. Phase Nine through to Phase Twelve are revisited), however, this must be fully authorised by the I·T Director.
Phase Deliverables: (Critical Items Only)	• Complete product archive, securely held. • Project documentation securely archived.

- This is the final phase of the life cycle model and concerns the ending of a product's life.
- The Product Manager needs to specify the various systems support and upgrade arrangements that are to be offered to the product's existing customers.
- Development needs to ensure that there exists a comprehensive archive of all the code and documentation associated with the product and then remove the product from any live services.
- Ideally archived systems needs to remain in a state where upon they can be readily reinstated as required; that is to say reverting back to Phase Ten: Production & Client Implementation.

Sample QA Documentation

There are two distinct types of documentation employed; these are Quality Management System Documents and Project Documents.

Quality Management System Documentation …

- The Quality Management System documentation consists of policies, procedures and working instructions that must be adhered to in order that the underlying processes remain stable, useable and productive.

- It is one of the QA Manager's main responsibilities to ensure that the Quality Management System is documented sufficiently and that the defined practices are being carried out effectively within the Development Teams and Software Testing / QA Teams. In effect conducting internal QA Audits.

- Although part of the role of the QA Manager is to be both constructively critical and to enforce QA standards, the role of the QA Manager should always be to assist, facilitate and co-ordinate the software testing and quality assurance. Thus empowering the various team members to deliver the necessary quality and correct functionality within the product.

- The QA Manager holds and maintains a complete set of all Quality Management System documentation.

Project Documentation …

- It is the responsibility of the Development Manager, through the medium of Project Plans, to define which documents are necessary and which are not.

- The QA Manger is responsible for ensuring that all the documentation conforms to the mandatory elements of the Quality Management System, however, the QA Manager is not responsible for the content.

- In reality most documents are not produced in isolation, the authors will spend significant time discussing and inviting comments from the intended readership. The extent of this informal process will depend very much on scope, importance and complexity of the documents.

- Project Documents are assigned to an Author and also ideally a designated Approval Authority (consisting of one or more persons and typically line management). Having written the document and arranged for it to be reviewed, it is the responsibility of the Document Author to obtain approval from the Approval Authority. Until approval is given then the document cannot be officially recognised within the Quality Management System, however, the Development Manager, QA Manager or Product Manager may give overriding approval as deemed necessary.

- There is one simple rule for document review, and that is that it should be reviewed by a subset of its intended readership to ensure that it is appropriate. The premise for which is that the people who are going to use the document are best placed to know if it satisfies their needs or not.

- It is not the objective to produce perfect English prose, however, it is just intended to ensure that the content is complete, correct and unambiguous; with no omissions, nor duplications, nor conflicts.

- The evolution of the software design commences from the User Requirements and terminates with the Program Specification. The intervening stages are constituted by the Functional Specification and the Design Specification. Hence the Program Specification is derived from the Design Specification, the Design Specification is derived from the Functional Specification and the Functional Specification is derived from the User Requirements. Each successive document elaborates upon the previous document upon which it was based.

- The following table identifies the primary documentation for each Phase of the Systems Development Life Cycle.

Phase	Phase Description	Associate Documentation
1	• Planning / Initial Requirements Gathering.	• Draft Business Plan / Project Plan. • Draft User Requirements. • Draft Functional Specification. • Draft Test Plan.
2	• Functional Definition / Refining of Requirements.	• Business Plan / Project Plan. • User Requirements. • Functional Specification. • Test Plan.
3	• Design.	• Business Plan / Project Plan. • User Requirements. • Functional Specification. • Design Specification. • Test Plan.
4	• Implementation.	• Business Plan / Project Plan. • User Requirements. • Functional Specification. • Design Specification. • Program Specification. • Test Plan.
5	• Component Test.	• Test Plan. • Test Report. (i.e. actual results)
6	• Integration & System Test.	
7	• Alpha Test.	
8	• Beta Test.	
9	• Release.	• Release Notes.

Document Library …

All Quality Management System documentation and all approved Project documentation should be filed by its author within the appropriate archive files or collective electronic repository; providing common access for all team members.

Document Specifications …

The following pages contain the generic documentation designs (in an itemised check-list format) that have been provided as a basic framework; they can then be elaborated upon as necessary within the Quality Management System. As with software development, it is envisaged that document development will also follow an incremental and cyclic nature. Thus starting from a basic framework, the documentation would be subsequently refined and further defined as time progresses, until it reaches its final form. The following documentation specifications are for a fully evolved Quality Management Systems. From a programmer's point of view, the main critical document is the program specification that provides a guide as to the tasks to be performed by the software under construction. If no functional or design specifications exist then the program specification can be formed from the user / staff manuals for the product under development / continuing-maintenance. For larger projects then full systems analysis and project management documentation may be employed such as SSADM and PRINCE.

SSADM v4:	**Eva, M., 'SSADM: v4 A User's Guide 2nd Ed', McGraw Hill, 1994.**
PRINCE v2:	**CCTA, 'Managing Successful Projects with Price 2', London: The Stationery Office. CCTA – Central Computer & Telecomm's Office (UK).**

Project Plan

Document:	• Combined project plan with business case.
Purpose of Document:	• To define the resourcing and scheduling of a project.
Written By:	• Product manager. • Development manager. • Development team.
Read By:	• All.
Approved By:	• IT Director.
Synopsis:	• Project description & major objectives. • Project assumptions. • Deliverables. • External dependencies. • Third party requirements. • Operational environment. • * Cost / benefit analysis. • * Return On Investment (ROI) analysis. • * Risk / consequence analysis. • * Legal / moral ramifications. • ** Multi-level component breakdown of work required to be undertaken. • ** Estimated costs and durations for each work activity involved. Used for project estimating techniques. • ** Resource requirements / work schedule (i.e. People, skills, hardware, software, etc). • ** Quality considerations (i.e. Type of testing to be employed). • ** Progress reporting. • Approach employed to systems development lifecycle (e.g. Waterfall or RAD prototyping). • Methodologies / techniques to be implemented (e.g. SSADM, JSD / JSP, BOOCH, etc). • Case tools / graphical designers to be used (e.g. Ms Visio or SSADM select, etc).

Please note:

* Denotes that item is often considered as part of the business case which is an optional component of the project plan.

** Denotes that item is often considered part of the methodologies being followed for the project management and systems analysis / design. For example Prince coupled with SSADM. In most cases though, these areas relate to the actual project management involved. For more details on the theory of project management then please see Lock, d., ‘Project Management 6th ed’, Gower, 1996.

User Requirements

Document:	• User Requirement.
Purpose of Document:	• To define system requirements as stipulated by clients, their users and other project stake holders (as appropriate).
Written By:	• Product manager. • Development manager. • Development team.
Read By:	• All.
Approved By:	• Product manager. • Development manager.
Synopsis:	• Reported by & owned by. • QA test criteria (verification & validation). • Type: either URC or NSR. • Assigned priority (time-frame for action). • Assigned criticality (importance rating). • Functional aspects of requirement (i.e. Such as the tasks to perform). • Associated non-functional considerations; i.e. … - storage - security - usability - installability - error recovery - environment - intended usage / downtime levels - load, performance and stress factors - compatibility - maintenance - packaging & media - training requirements - support requirements • Cost and benefit analysis for requirement (as appropriate). • Cross-reference to all documents and artefacts that are related to the requirement. • Cross-reference of other requirements that are related to the requirement. Those that are dependant are highlighted so that the knock-on effects of change can be maintained.

There are many different approaches to the specification of User Requirements ranging from the simplistic SMART (Specific, Measurable, Attainable, Relevant and Timely) through to those that are multifaceted and criterion based with comprehensive QA controls. An example of criterion based User Requirements speciation is discussed on the next page.

User Requirements (Continued)

The user requirements should form the foundations upon which any computer system is built and as such they should be subject to appropriate quality guidelines such as the following compliance criteria as recommended within pages 407 to 416 of Dustin, Rashka and Paul, 'Automated Software Testing', Addison-Wesley, 1999.

- Criteria #1: Each requirement must have a quality measure (either Qualitative or Quantitative) that can be used to test whether any solution meets the requirement.
- Criteria #2: The specification for the requirement must contain a definition of the meaning of every essential subject matter or term used to define the requirement.
- Criteria #3: Every reference to a defined term must be consistent with its definition.
- Criteria #4: The context of the requirement must be wide enough to cover everything that is needed to be understood.
- Criteria #5: The stakeholder(s) must be asked to disclose and prioritise their known current and future requirements (i.e. 'wants' & 'needs') so that these can be considered accordingly within the design process. Particularly looking for those requirements that if brought into the main arena would potentially derail or at least temporally destabilise the project. Typical examples being people's private agendas and personal wish-lists that are rarely openly discussed and yet are often imposed by stealth, subterfuge and other underhand tactics.
- Criteria #6: Every requirement in the specification must be relevant the system.
- Criteria #7: The specification must NOT contain requirements that are posturing as solutions. A requirement must represent a genuine required resolution to a constraint or a need. A requirement should not dictate how the constraint or need will be resolved.
- Criteria #8: The stakeholder(s) must attribute defined values for each requirement so that their worth and timescales for implementation can be evaluated and prioritised accordingly.
- Criteria #9: Each requirement must be uniquely identifiable for further recall and reference.
- Criteria #10: Each requirement must be tagged to all parts of the system where it is used so that it can be cross-referenced. Thus for any change to the system, or one of its requirements, then there will be appropriate pointers that identify all the other parts of system, including associate requirements, where the changes will have effect.
- Criteria #11: Where appropriate, requirements should be time-framed and associations to stake-holders made known.

Finally, user (and system) requirements can be categorised as either User Requirements for Change (i.e. URC) and New Systems Requirements (i.e. NSR), both of which must be fully approved by management before implementation. Both the User (and System) Requirements provide the primary inputs to the Functional Specification. A record of their evaluation should be retained confirming their approval (or rejection) and assigned prioritisation according to the MoSCoW rules …

(M)ust Do / (S)hould Do / (C)an Do / (W)ont Do

IEEE 820 is the standard for SRS (software requirements specification). Essentially it matches with the above criterion and indeed IEEE 820 elaborates further by defining in detail the characteristics of a good SRS as being Correct, Unambiguous, Complete, Consistent, Ranked for importance and / or stability, Verifiable, and Modifiable. IEEE 820 is discussed in detail within its own specific briefing.

Functional Specification

Document:	• Functional specification.
Purpose of Document:	• To define the functionality and the operation of the product from the user's perspective. • To act as the main reference for input to the design and implementation phases.
Written By:	• Product manager. • Development manager. • Development team.
Read By:	• All.
Approved By:	• Product manager. • Development manager.
Synopsis:	• High-level narrative description of the functionality to include the envisaged software capabilities. • Cross-reference of functionality against user (and system) requirements and also cross-referenced against other systems analysis and design documentation. • Background publications and discussion documents referenced within system design. • Intended storage. • Intended security. • Intended usability. • Intended installation procedures & requirements. • Intended error recovery. • Intended operational environment. • Intended usage / downtime levels • Intended load, performance and stress factors. • Intended compatibility. • Intended maintenance. • Intended packaging and media. • Intended training requirements. • Intended support requirements.

There are various methodologies that can be employed within the Systems Analysis and Design documentation which includes SSADM, JSD / JSP and BOOCH amongst many others.

Of these SSADM (or one of its many variants) is commonly used to graphically depict and document the Systems Analysis and Design using Data Modelling, Business Process Modelling and Sequence of Operation Modelling techniques.

SSADM v4: Eva, M., 'SSADM: v4 A User's Guide 2nd Ed', McGraw Hill, 1994.

Ideally the Functional Specification should directly integrate with the Systems Analysis and Design documentation in order to provide the programmer with full scope and context of the system being developed, however, to be of actual benefit the programmer must be able to interpret the methodology being employed. All too often the Systems Analysis and Design is kept within the domain of the Systems Analyst, Business Analyst and Project Manager and not distributed in an appropriate form to those who actually develop and test the code. As a result many systems are doomed to failure even before they even begin; due simply to a lack of understanding of HOW and WHY it should work!

Design Specification

Document:	• Design specification.
Purpose of Document:	• To record the overall function and structure of a software program in a manner that coding may proceed more accurately and efficiently. • To record cross-references to the test strategies, test cases, sample data and expected results, etc as detailed within the corresponding test plans. • To record cross-references to the relevant sections of the corresponding project plans. • To record the structure and function of the build and install process. • To simplify subsequent maintenance.
Written By:	• Development team
Read By:	• All.
Approved By:	• Development manager.
Synopsis:	• Mid-level narrative description of the functionality to include the envisaged software capabilities.

This can be totally free form and may be presented in any format that achieves the required purpose e.g. Using a graphical CAD product or CASE toolset. If special tools are used, then a version of the specification of the product should be available that is accessible to those who have not purchased the product.

A pro-forma for a design specification should contain the following basic items:

- Title.
- Abstract.
- Programming language & environment.
- Menu structures and screen dialogues.
- Additional inputs files / devices (if appropriate).
- Additional output files / devices (if appropriate).
- Random access files (if appropriate).
- Database schemas and sub-schemas (including entity-relationships & file structures).
- Report layout definitions and sample printouts.
- Description of query extractions / data processing performed using narrative scripts. Includes relevant SQL, pseudo code and flowcharts / diagrammatic representations as deemed necessary for efficient and effective development of software.
- Cross-references to corresponding project plans, test plans and functional specification.
- Error handling and exemption reporting.
- Also relevant would be a list of sub-components that can be reused from other projects or purchased from third-party product vendors. In addition documented protocols for software development and pre-defined ‘program-coding’ templates and code libraries should also be specified for a ‘copy & paste’ approach to development.
- References to technical documentation and publications that details the underlying mechanisms and theory regards the intended system operation. For example when implementing Financial systems then various Accountancy and Book-Keeping texts would be consulted.

NB: The design specification provides primary input to the program specification. ‘Pictorial Storyboard & Bullet-Point Annotations’ and ‘Freestyle Narrative Script’ techniques are ideal mechanisms for presenting a paper based prototype of how the system is intended to work covering menus, screen dialogues and data processing. In addition a flow-graph / flow-chart diagrams can also be used to depict the sequences and logical conditions that will affect the execution paths.

Program Specification

Document:	• Program specification.
Purpose of Document:	• To further define and refine upon the design specification and to provide the main reference for the update of the design specification and appropriate system documentation. • To act as the working document used as the main reference while developing the program code.
Written By:	• Development team.
Read By:	• All.
Approved By:	• Development manager.
Synopsis:	• Low-level narrative description of the functionality to include the envisaged software capabilities.

Constituted from the content of the functional specifications and the design specifications; these would be combined and expanded to form the program specifications. The expansion would include the internal logic specified using FlowGraphs and Pseudo Code as appropriate. The Program specification would be rationalised accordingly to remove any duplications, omissions and contradictions.

Although it is quite often that case that the functional specification and the design specification are omitted and only the program specification is employed. Indeed in these circumstances, the program specification is often barely little more than a discussion document detailing functionality that has usually been drafted from the original system / user manuals and is primarily maintained by the programmer. Upon completion the design specification can then be subsequently reconstituted back into the system / user manuals to maintain documentation integrity; hence if required the manuals can act as the formal system specification. From experience, this simple approach can be astonishingly effective requiring only the minimum of effort and time. For more details see the section entitled “Author’s notes on compiling High Level testing scripts”.

Test Plan (incorporating IEEE 829 & BS 7925)

Document:	• Test plan.
Purpose of Document:	• Defines the test methods and exit criteria to be used. • Reference to ensure that adequate testing is done.
Written By:	• Development manager / QA manager. • Development team / QA team.
Read By:	• All.
Approved By:	• QA manger.
Synopsis:	• Planned Component testing. • Planned system & integration testing. • Planned alpha testing. • Planned beta testing. • Planned documentation checking. • Planned final testing.

Ideally the Test Plan is constituted using an industry standard format such as the IEEE 829-1998.

Framework of the IEEE 829 – 1998 Standard for Software Test Structure

Test Plan ...

- 4.2.1 Test plan identifier (including level designation);
- 4.2.2 Introduction;
- 4.2.3 Test items;
- 4.2.4 Features to be tested;
- 4.2.5 Features not to be tested;
- 4.2.6 Approach;
- 4.2.7 Item pass / fail criteria;
- 4.2.8 Suspension criteria and resumption requirements;
- 4.2.9 Test deliverables;
- 4.2.10 Testing tasks / procedures;
- 4.2.11 Environmental needs;
- 4.2.12 Responsibilities;
- 4.2.13 Staffing and training needs;
- 4.2.14 Schedule;
- 4.2.15 Risks and contingencies;
- 4.2.16 Approvals.

Test design specification ...

- 5.2.1 Test design specification identifier;
- 5.2.2 Features to be tested;
- 5.2.3 Approach refinements;
- 5.2.4 Test identification;
- 5.2.5 Feature pass / fail criteria.

Test case specification ...

- 6.2.1 Test case specification identifier;
- 6.2.2 Test items;
- 6.2.3 Input specifications;
- 6.2.4 Output specifications;
- 6.2.5 Environmental needs - Hardware / Software / Other;
- 6.2.6 Special procedural requirements;
- 6.2.7 Intercase dependencies.

Test procedure specification ...

- 7.2.1 Test procedure specification identifier;
- 7.2.2 Purpose;
- 7.2.3 Special requirements;
- 7.2.4 Procedure steps - Log / Set up / Start / Proceed / Measure / Shut down / Restart / Stop / Wrap up / Contingencies;

Test item transmittal report ...

- 8.2.1 Transmittal report identifier;
- 8.2.2 Transmitted items;
- 8.2.3 Location;
- 8.2.4 Status;
- 8.2.5 Approvals.

Test log ...

- 9.2.1 Test log identifier;
- 9.2.2 Description;
- 9.2.3 Activity and event entries - Execution description / Procedure results / Environmental information / Anomalous events / Incident report identifiers.

Test incident report ...

- 10.2.1 Test incident report identifier;
- 10.2.2 Summary: Summarise the incident;
- 10.2.3 Incident description - Inputs / Expected results / Actual results / Anomalies / Date and time / Procedure step / Environment / Attempts to repeat;
- Testers / Observers;
- 10.2.4 Impact.

Test summary report ...

- 11.2.2 Summary;
- 11.2.3 Variances;
- 11.2.4 Comprehensiveness assessment;
- 11.2.5 Summary of results;
- 11.2.6 Evaluation;
- 11.2.7 Summary of activities;
- 11.2.8 Approvals.

The framework of the IEEE 829 – 1998 Standard for Software Test Structure as outlined above can be applied to all levels of testing (i.e. Component, System & Integration, Alpha, Beta and Final).

Indeed where appropriate, the items on the on the Test Plan (also referred to as a QA Plan) can be merged as necessary. Furthermore to minimise the content of the Test Plan, it is viable to establish a common policy guide that is assumed to be followed in all projects and referenced within the Project Plan. This policy guide would consist of ‘standard instructions’ that apply to all projects and may include that the methodology followed for the software testing and quality assurance, such as BS7925 for the Component testing. For example see briefing on employing a generic QA Plan for IEEE 829 implementation. Likewise, reference can be made to specific test scripts retained within suitable libraries on a central repository to avoid duplication. Thus the Test Plan itself may simply be constituted as a historical reference record identifying who undertook the testing, when it was undertaken, a general description of the approach followed (including references to test scripts) and brief report as to the outcome. See briefing on Test Catalogues.

Framework of the BS7925 – 2001 Standard for Software Test Methods

Part One: Terminology …

- *Component (5.43):* A minimal software item for which a separate specification is available.
- *Component Testing (5.44) also known as Module Testing or Unit Testing (5.213):* The testing of individual software components.

  A repeat of Component Testing using the same test script is referred to as a Retest.
- *Integration Testing (5.114):* Testing performed to expose faults in the interfaces and in the interaction between integrated components.
- *System Testing (5.187):* The process of testing an integrated system to verify that it meets specified requirements.
- *Regression Testing (5.154):* Retesting of a previously tested program following modification to ensure that faults have not been introduced or uncovered as a result of the changes made.

  Regression test scripts are constituted from the re-use of component tests and system & integration tests. Alternatively regression test scripts are constituted from both component tests and system & integration tests which have been combined together to form a collective script and rationalised accordingly to remove any duplications, omissions and contradictions.
- *Acceptance Testing (5.1):* Formal testing conducted to enable a user, customer, or other authorised entity to determine whether to accept a system or component.
- *Project Mandate & Schedules:* An umbrella document providing the authorisation and agreed test plan that encompasses component testing, retesting, system & integration testing, regression testing and acceptance testing. An ideal format is IEEE829, the industry standard for software test specification. It would also include reference to stake holder signatories, cost estimates and planned timescales.

Part Two: Test case design and measurement techniques …

- Equivalence Partitioning.
- Boundary Value Analysis.
- State Transition Testing.
- Cause-Effect Graphing.
- Syntax Testing.
- Statement Testing.
- Branch / Decision Testing.
- Data Flow Testing.
- Branch Condition Testing.
- Branch Condition Combination Testing.
- Modified Condition Decision Testing.
- LCSAJ Testing*.
- Random Testing.
- Other Testing Techniques.

*LCSAJ - Linear Code Sequence and Jump

Release Notes

Document:	• Release notes.
Purpose of Document:	• To present the users and staff with information about the product that is different from, or additional to, that contained in the product manuals, e.g. Lists of reported problem fixed by the release, etc.
Written By:	• Development team.
Read By:	• All.
Approved By:	• Development manager.
Synopsis:	• Totally product dependant. A project aim should be to minimise the content of the release notes since they only contain non-standard information.

The level of release documentation is dependent upon the organisation. It can range from the bare minimum through to a comprehensive explanation of the system variation / new features included within the release. Although not mandatory it is advisable to include references to the corresponding specifications, user / staff manuals, installation instructions and appropriate testing procedures that can be followed as part of clients' User Acceptance Testing (UAT).

Sample QA Review Check List

There are several levels of review, these are; (1) **Informal peer reviews**, (2) **Walkthroughs**, (3) **Formal technical reviews** and (4) **Formal inspections.** Essentially these reviews become more formalised and disciplined as progression is made through the Systems Development Lifecycle and the various layers are navigated in a top-down fashion. Whereas the upper levels are informal and peer based, the lower-levels involve participants at various levels of the organisation with proceedings being officially documented and supervised by a designated review leader or chairman.

The most common type of review is the Walkthrough, for which Roger Pressman defined a useful checklist for use by development and testing teams alike:

System Engineering …

- Are major functions defined in a bounded and unambiguous fashion?
- Are interfaces between system elements defined?
- Have performance boundaries been established for the system as a whole and for each element?
- Are design constraints established for each element?
- Has the best alternative been selected?
- Has a mechanism for system validation and verification been established?
- Is there consistency among all system elements?

Software Project Planning …

- Is software scope unambiguously defined and bounded?
- Are resources adequate for the project scope?
- Is terminology clear?
- Are resources readily available?
- Have risks in all-important categories been defined?
- Is a risk management plan in place?
- Are tasks properly defined and sequenced?
- Has parallelism been taken into account ensuring necessary resources are allocated?
- Is the basis for cost estimation reasonable?
- Has the cost estimate been developed using two or more independent methods?
- Have differences in the estimation methods been reconciled?
- Is the planned schedule consistent?

Software Requirements Analysis …

- Is information domain analysis complete, consistent and accurate?
- Is problem partitioning complete?
- Are external and internal interfaces properly defined?
- Does the data model properly reflect data objectives, their attributes and relationships?
- Does the data model accommodate the implementation of the intended system functionality?
- Are all requirements traceable to system level?
- Is prototyping traceable to system level?
- Is performance achievable within the constraints imposed by other system elements?
- Are requirements consistent with schedule, resources and budget?
- Are validation criteria complete?

Software Design (Preliminary) …

- Are software requirements reflected in the software architecture?
- Is effective modularity achieved?
- Are components functionally independent?
- Is the program architecture factored?
- Are interfaces defined for components and external system elements?

- Is the data structure consistent with the information domain?
- Is the data structure consistent with the software requirements?
- Has maintainability been considered?
- Have software quality factors been explicitly expressed?
- Do the various algorithms accomplish the desired function?
- Are the algorithms logically correct and formulae validated?

Software Design (During Development)…

- Does the algorithm accomplish the desired function?
- Is the interface consistent with the architectural design?
- Is the size and complexity of the logic reasonable?
- Have error handling and 'anti-bugging' been specified?
- Are local data structures properly defined?
- Are structured programming constraints used throughout?
- Is design detail amenable to implementation language?
- Are operating system and language dependent functions documented?
- Is the compound or inverse logic documented?
- Are the technicalities and practicalities of the design explained?
- Has maintainability been considered?

Software Coding …

- Has the design properly been translated into code? Where appropriate the program design and associate documentation should be available during this review. (i.e. flow charts, flow graphs or JSP, etc)
- Are there misspellings and typos?
- Has proper use of language conventions been made?
- Are there any incorrect or ambiguous comments?
- Is there compliance with coding standards for language style, comments and component prologue?
- Are the data types and data declarations in the proper format and syntax?
- Are physical constants correct? Are they used appropriately?
- Have all the items on the design walkthrough checklist been reapplied to the code as required?

Software Testing (Test Plans)…

- Have major test phases properly been identified and sequenced?
- Has traceability to validation criteria / requirements been established as part of software requirements analysis?
- Are major functions demonstrated early?
- Is the test plan consistent with the overall project plan?
- Has a test schedule been explicitly defined?
- Are test resources and tools identified and available?
- Has a test record-keeping and analysis mechanisms been established?
- Have 'stress' and 'load' testing for software been specified? For example, operational extremity testing and multi-user operation / network performance.

Software Testing (Test Procedures) …

- Have both white box and black box tests been specified?
- Have all the independent logic paths been tested within white box testing?
- Have all test cases been identified and listed with their expected results?
- Is error handling to be tested? If so, how?
- Are boundary values to be tested? If so, how?
- Are timing and performance to be tested? If so, how?
- Has an acceptable variation from the expected results been specified where applicable?

Software Maintenance …

- Have side effects associated with change been considered?
- Have requests for change been documented, evaluated and approved?
- Has the change, once made, been documented and reported to all interested parties?
- Has a final acceptance review been conducted to ensure that all software components have been properly updated, tested and replaced as appropriate?

Pressman, Roger S., 'Software Engineering – A practitioner's Approach', McGraw Hill 1994.

Once a QA risk assessment has been completed then the mitigation action plan can be implemented, which can be as simple as defining What-Who-When-Where-How narratives or employing more formalised procedures and documentation.

Case Study in employing a High Level approach to Testing

For this case study, a sample snippet of program code will be analysed; the purpose of the function is to enter up to 100 numbers until either the 100th number is entered or a rogue value of –999 is entered. After which the calculated results are displayed. For example, the procedure 'Average' may be invoked from code or directly from the immediate debug environment using Call Average [1,2,3,4,5, -999,500,0]. It would display the value of 3 on screen (for the Average) and return the value of 3 for the Calculate Average, 15 for the Input Total and 15 for Valid Total. **Throughout the examples presented will be based upon Fitzsimmons, A. and Love, T., 'A Review and Evaluation of Software Science', ACM Computing Surveys, vol. 10, No 1, March 1978; duly expanded using the ISEB Foundation Certificate Syllabus for Software Testing.**

The high level approach presented here is 'black-box' in nature; where the emphasis is specifically placed upon the direct inputs and outputs of the program (or program component) that is under test. There is little or no regard as to the underlying logic, which in theory should be completely encapsulated, and so is beyond the scope of the testing itself. As a result it would be true to say that not all execution paths and operational criteria could be readily identified by using such a high level approach; indeed in some circumstances this level of testing may not even be attainable. However, it may be decided whist compiling the test procedures that the high-level test procedures would need further derivation to lower levels. This is particularly the case if the functions being tested are considered mission-critical, risky, large or complex. This is covered in the later section entitled 'Case Study in employing a Low Level approach to Testing'.

Upon inspection of the PDL (listed at the end of this section), it can be determined that there are seven execution paths for which the corresponding Test Cases have been compiled. Test Cases are also known as known as 'Usage Scenarios' or 'Stories'.

Test Case 1: The function should abort at commencement upon entering a rouge value of -999. The results returned would be invalidated.

Test Case 2: The function should abort at mid-operation upon entering a rough value of -999. Initially a value of 1 is entered to commence the operation and this is followed by the -999 terminating value. The results returned should be verified using a calculator or checked against a spreadsheet containing suitable test inputs and expected test results.

Test Case 3: The function should allow entry of more than 100 entries using all invalid values, however, it should not allow the 101st entry at that point as it should terminate automatically. All the values entered would either be less than the minimum value (i.e. 0) or greater than the maximum value (i.e. 500) so that they would be counted as invalid. The results returned should be verified using a calculator or checked against a spreadsheet containing suitable test inputs and expected test results.

Test Case 4: The function should allow entry of more than 100 entries using some or all valid values, however, it should not allow the 101st entry at that point as it should terminate automatically. Some of the values entered should either be less than the minimum value (i.e. 0) or greater than the maximum value (i.e. 500) so that they would be counted as invalid, however, some values should be valid. The results returned should be verified using a calculator or checked against a spreadsheet containing suitable test inputs and expected test results.

Test Case 5: The function should allow up to 100 entries using all valid values, however, before the 101st entry a rouge value of -999 should be entered so that the function terminates. All of the values entered should be within the boundaries of minimum value (i.e. 0) and maximum value (i.e. 500) so that they are valid. The results returned should be verified using a calculator or checked against a spreadsheet containing suitable test inputs and expected test results.

Test Case 6: The function should allow up to 100 entries using some or all valid values, however, before the 101st entry a rouge value of -999 should be entered so that the function terminates. Some of the values entered should be greater than the maximum value (i.e. 500) so that they would be counted as invalid. The results returned should be verified using a calculator or checked against a spreadsheet containing suitable test inputs and expected test results.

Test Case 7: The function should allow up to 100 entries using some or all valid values, however, before the 101st entry a rouge value of -999 should be entered so that the function terminates. Some of the values entered should be less than the minimum value (i.e. 0) so that they would be counted as invalid. The results returned should be verified using a calculator or checked against a spreadsheet containing suitable test inputs and expected test results.

Assumptions: Minimum valid value is 0. Maximum valid value is 500.

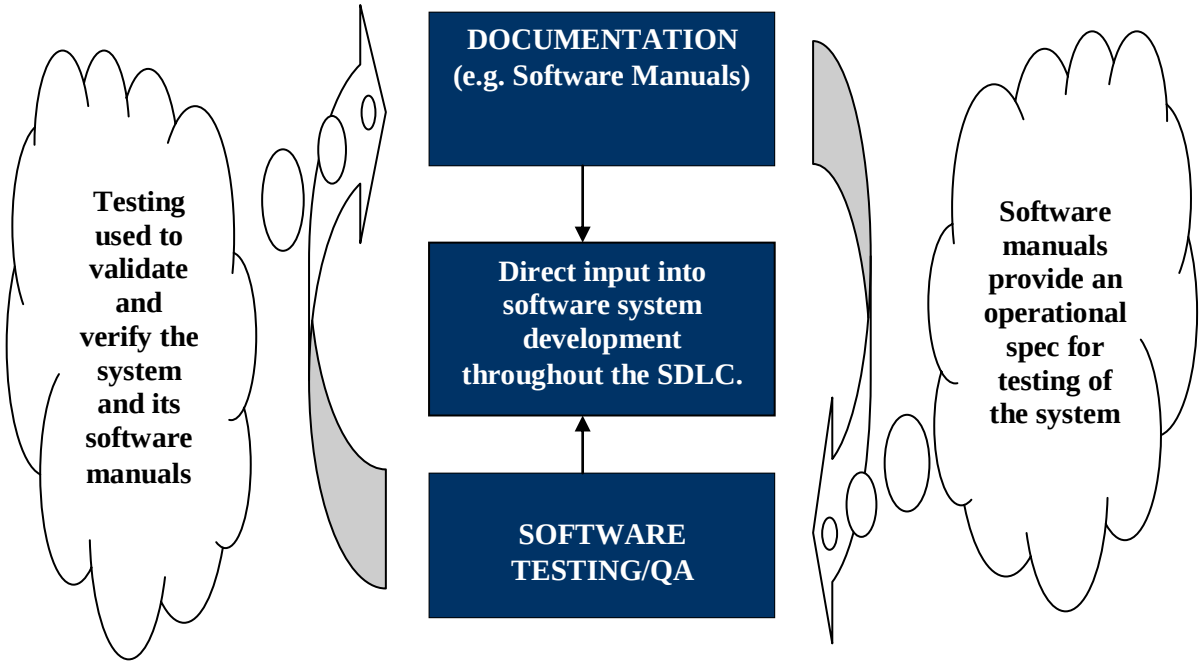
PDL / Program Code Listing:

```
PROCEDURE average
REM this procedure computes the average of 100
REM or fewer numbers that lie between bounding
REM values; it also computes the sum
REM and total number valid.
PROCEDURE ACCEPTS value AS INTEGER ARRAY [1:100]
PROCEDURE ACCEPTS maximum AS INTEGER
PROCEDURE ACCEPTS minimum AS INTEGER
PROCEDURE RETURNS average AS INTEGER
PROCEDURE RETURNS total_input AS INTEGER
PROCEDURE RETURNS total_valid AS INTEGER
DEF i AS INTEGER
DEF sum AS INTEGER
LET i = 0
LET sum = 0
DO WHILE value[i] <> - 999
AND total_input < 100
        LET total_input = total_input + 1
        IF value[i] >= minimum
        AND value[i] <= maximum
        THEN
                LET total_valid = total_valid + 1
                LET sum = sum + value[i]
        ELSE
                REM – Skip (i.e. perform no operation)
        END IF
        LET i = i + 1
REPEAT DO
IF total_valid > 0
THEN
        LET average = sum ÷ total_valid
ELSE
        LET average = -999
END IF
DISPLAY average
END PROCEDURE
```

NB: The above example has also been applied within the subsequent Case Study in regards employing a Low Level approach to Testing; in which the same exercise is repeated to a greater depth of detail with more explanatory notes.

Author's notes on compiling High Level testing scripts

From experience, the easiest approach to writing High Level testing scripts is to employ a heuristic approach in which the focus is placed upon establishing a mutually beneficial symbiotic relationship between documentation and testing that is both cost-effective and efficient. Where by the deliverables from one activity constitutes the inputs for the other and so an incremental cycle is established, in which testing and documentation are defined and further refined throughout the systems development project.



HINT: The construction of the test scripts could be commenced by implementing the following technique. For each program, identify every system feature within its specification, and then for each feature identify every data field, runtime user input and function employed; identifying every possible value (as appropriate). From this definition of the system and its boundaries you can then construct a test case to exercise every combination of program, function, field and corresponding input value covering both 'positive' (+ve) testing and negative (-ve) testing.

HINT: Alternatively when having difficulties in constructing test scripts, it always easier to start with a 'source of reference document' that contains every relevant excerpt that can be sourced from the various specifications and / or technical documentation regarding the new or changed functionality. This can be reworked and added to accordingly to provide a 'handle' or 'insight' from which the testing strategy can be evolved through an iterative and incremental process. It also enables others to see where the test scripts originated from during their construction; hence it is necessary to include cross-references as appropriate.

In all cases, the starting point when compiling a high level test script is to construct the initial highest-level narrative procedure to be followed that focuses upon the method undertaken. This literally consists of one or more paragraphs (or bullet points) that describe the strategies and tasks to be undertaken within the test procedure and normally follows the 'black-box' approach to testing. This may then be supplemented by additional Narrative Text and Tabulated Grids which are used for various purposes; including test instructions, benchmarked test data and expected test outcome, etc.

After the first level of Functional Testing is specified the question of ***'Is this testing procedure adequate for its intended task?'*** is asked. Within the deliberations a number of criteria may be considered, however, the two most important of which are concerned with the robustness and coverage of testing:

- ***Is the robustness of the test procedure able to withstand reasonable scrutiny by both internal and external parties? This would include …***

    - Customers and their End Users (particularly those conducting UAT)
    - Management (including Project Stakeholders)
    - Staff (including Support, Sales and Development)
    - Lawyers (particularly the clients legal representative)
    - QA Auditors (including external inspectors & regulatory bodies)
    - And so on

    NB: It is important that the testing is presented in format that is most suitable for its intended readership and that can be correctly interpreted at a future point which may be several months or even several years later!

- ***Could the testing be expanded further and if so does it really require further levels of derivation?*** Alternatively a more clinical approach to this question would *be 'Has the exit criteria and appropriate quality standards been met?'* i.e. such as …

    - Budgets
    - Deadlines
    - Number, type and severity of incidents identified
    - Number, type and severity of incidents still outstanding
    - Number, type and severity of incidents successfully resolved
    - Depth of testing has reached the predetermined level
    - And so on

Other criteria can be established, however, the above two items provide the crux of the decision making process within a typical 'heuristic' approach.

If it is considered that the current level of testing for the test procedure is not sufficient, then the test procedure is further defined and refined to the next lower level. After which the question *of* ***'Is this testing procedure adequate for its intended task?'*** is repeatedly reviewed until no further refinement (from the higher to lower levels of testing) is considered necessary.

Thus an iterative and incremental cycle is undertaken that defines and further refines the testing strategies and procedures, in effect a RAD prototyping technique for software testing is being employed that targets the functionality of the system. The test scripts are literally evolved and expanded to the level deemed appropriate given the circumstances at the time. Initially targeting test coverage in terms of breadth before proceeding on to targeting test coverage in terms of depth.

Author's notes on the Complexities of Scale and Test Coverage

The issue of software size and complexity in relation to testing is very important and often forms the crux of the many issues faced within IT delivery. Far too many managers and software developers still implement software testing in a 'half-baked' approach, or worse still, completely reject software testing as being unworkable and impracticable. This is particularly the case for many large Government and Corporate projects, as well as, frequently being rife within software development within the Commercial sector. It is true to say that off-the-shelf products are less affected as most software providers have their own organisational requirement for their products to be stable and functional so that they can be successfully supported and marketed. This is particularly the case where there is a sizeable client base and where there are several rival competitors operating within the same business sector. Indeed such companies may even participate in industry standards, such as ISO 9001:2000, for these provide quality assurances to clients and also provides a marketing advantage.

The harsh commercial realities being that software testing is neither simple, quick nor cheap; however, this just means that some sort of rationalisation of the testing process will have to be implemented.

As an analogy, imagine an Oak tree as representing software system under test, where by the trunk would represent the program being tested, the branches and twigs being the execution paths and the leaves being the resultant conditional outcomes. With a sapling grown from an Acorn, it is possible to view all aspects of the structure from its trunk through to its individual leaves, whereas with a mighty Oak, often its internal structure is obscured by its outer layer of leaves and so it is often beyond manageable comprehension. It is therefore essential to specify the scope and boundaries within which to focus the effort, so that the testing can be targeted efficiently and effectively through appropriate planning and prioritising. Hence the question '***Could the testing be expanded further and if so does it really require further levels of derivation?***' should be asked to determine the level of detail (i.e. scope & boundaries)

Furthermore, one should look to maximise the breadth of coverage in order to test as much of the functionality as possible, however, the depth of testing would be dependent upon the type of function being tested and the current stage within the development and testing process. Hence ideally each function would be analysed to determine the priority and level of testing required and the construction of the test procedures would be an incremental and iterative process consisting of several repetitions where by all of the testing procedures are expanded collectively as a batch so that they are matured at the same rate. They would commence from a high level of definition and this would be repeatedly refined and further defined to lower levels of definition. This means that no test procedure should remain immature and that the cyclic expansion of a test procedure only stops when the level of testing required has been reached (i.e. its intended depth of testing has been attained), however, as to be expected some test procedures will have been expanded more than others. For example, the testing required to verify and validate a complex analysis report would be considerably more comprehensive than the testing required to verify and validate a generic enquiry screen. Hence the question '***Is the robustness of the test procedure able to withstand reasonable scrutiny by both internal and external parties?***' should be asked to determine whether further expansion of the test scripts is required and if so which appropriate areas of the system should be targeted (i.e. quality testing and not just quantity testing).

.

Author's notes on Priority Allocation Techniques

Where it is considered that Priority Allocation is required for test planning then the selection criteria for determining the test candidates may be undertaken using the following approach:

- Criticality of the candidate function / feature / operation. This is a prioritisation-based assessment where by functions are gauged as to how they support the mission objectives of the application.

- Usage level of the candidate function / feature / operation. This is a usage frequency based assessment where it is assumed that those functions that are most used should be tested with more emphasis than those that are used less often.

- Complexity and Size of the candidate function / feature / operation. This is an effort-based assessment where it is assumed that the more complex functions will require more effort and so precedence should be applied to these functions during the testing process.

- Regularity of change that is expected for the candidate function / feature / operation. Some functions are likely to be updated and enhanced more often compared to others, this factor should be considered as these will need repeated testing in order to take in to account any changes that are made.

- Risk that the candidate function / feature / operation will contain unseen bugs or errors. This is a likelihood-based assessment of the influence of external and internal factors, where the severity and frequency of the risk is included within the estimation (i.e. potential exposure). Internal factors would include size of project coding, programmer's experience / ability, development environment, programming language / database characteristics, team-dynamics, budgets and deadline constraints, technical practicalities etc. External factors would include management directives and support desk issues, company policy, legal obligations, expected return on investment (ROI) and designated time to market, etc.

- Minimum level of quality assurance required to test the function / feature / operation. Not every function needs the same level of testing; some need less, while others need more.

- Impact that the candidate function / feature / operation would have if it failed. In other words the cost and consequences of failure should be clearly defined and made clear to everyone at all levels. Ideally the impact of system failure should be categorised in hierarchical levels such as (Level 1) catastrophic failure, (Level 2) non-catastrophic but still not an acceptable failure, (Level 3) non-catastrophic but an acceptable failure and (4) general software annoyance or teething problem.

NB: The term 'function / feature / operation' is simply a case of terminology semantics.

Quantifying such criteria can either be conducted by intuitive assessment or by formal metrics as demonstrated here …

- A list of application specific functions / features / operations is compiled (usually from a Test Catalogue) and those that were Human Computer Interface (HCI) oriented would be removed from the list. Hence leaving only the Data Processing oriented; i.e. such as the Unattended Batch Operation (UBO) functions. Both HCI and UBO functions are discussed within the section entitled 'Briefing on Semi-Automated & Inbuilt Testing Techniques'.

- Each of the remaining functions / features / operations would be prioritised using the above criteria for Criticality, Usage Level, Complexity, Regularity of Change and Risk. Before they are prioritised, they would first be assigned to Impact levels 1 to 4, where by those functions in level 1 were tested according to their priority before those within level 2 and in the same context level 2 before level 3, and level 3 before level 4. This technique would ensure that each function / feature / operation would be tested according to both its significance (i.e. priority ranking) and its potential impact (i.e. those with highest potential impact category are tested first).

- The assessment for priority ranking could be conducted using a simple weighted feature point method where by each candidate function / feature / operation is assigned appropriate values for each of the criteria. The identifiers (and corresponding benchmark values) would be Insignificant (i.e. value of 1), Low (i.e. value of 3), High (i.e. value of 7) and Significant (i.e. value of 9). It was preferred to assign identifiers to criteria rather than direct values, as values are open to criticism and question, whereas identifiers are simple to understand and are easier for others to accept. Furthermore each of the criteria would be assigned corresponding weightings to highlight the appropriate significance, again using the same identifiers, however, they would be consistent for all candidate functions / features / operations.

  For example, the priority ranking for a Production Analysis report might be calculated thus …

  Impact of the Production Analysis failing would be considered as Level 2 Non-catastrophic but still not an acceptable failure and so would be tested only when all of the Level 1 functions were complete in their testing and before levels 3 and 4 are commenced.

  The priority rank for the test procedure would be estimated as follows …

*** Priority Rank for Production Analysis ***			
	Criteria Weighting (Fixed)	**Function / Feature / Operation Weighting (Variable)**	**Line Total**
Criticality	H (i.e. 7)	H (i.e. 7)	7 * 7 = 49
Usage Level	S (i.e. 9)	L (i.e. 3)	9 * 3 = 27
Complexity / Scale	H (i.e. 7)	S (i.e. 9)	7 * 9 = 63
Regularity of Change	L (i.e. 3)	I (i.e. 1)	3 * 1 = 3
Risk	H (i.e. 7)	L (i.e. 3)	7 * 3 = 21
Grand Total	N/A	N/A	49+27+63+3+21 = 163

  Hence the Priority Rank for the above Production Analysis is 163 (within Impact level 2).

- The use of metrics in test procedure selection is strongly advocated by many in software testing, for an in depth appraisal of such techniques please see the paper published by Hans Schaefer of 'Professional Tester' magazine at http://c2i.net/schaefer/testing/risktest.doc

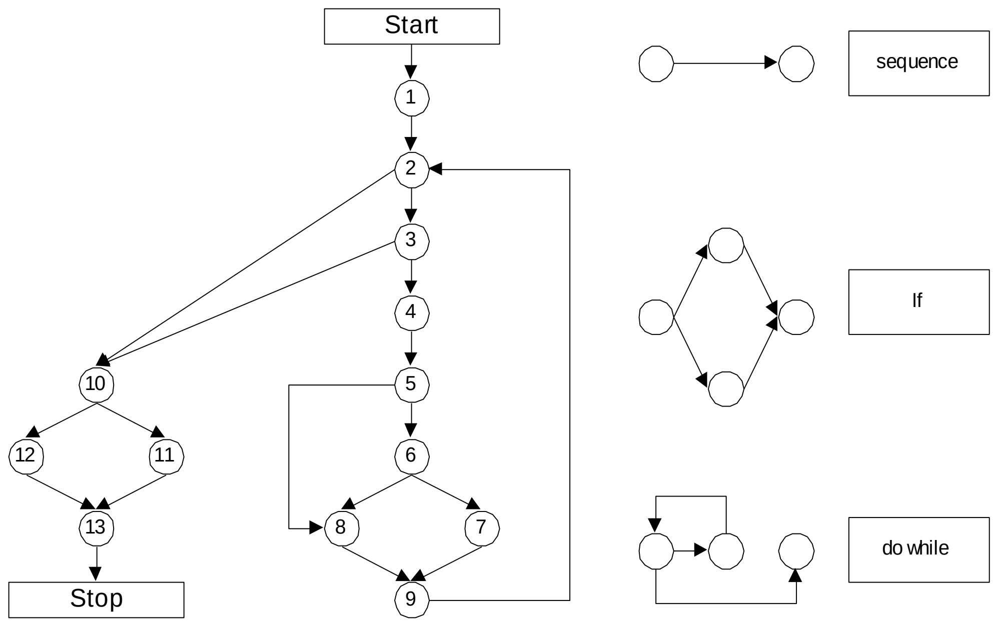
Case Study in employing a Low Level approach to Testing

For this case study, a sample snippet of program code will be analysed; the purpose of the function is to enter up to 100 numbers until either the 100th number is entered or a rogue value of –999 is entered. After which the calculated results are displayed. For example, the procedure 'Average' may be invoked from code or directly from the immediate debug environment using Call Average [1,2,3,4,5, -999,500,0]. It would display the value of 3 on screen (for the Average) and return the value of 3 for the Calculate Average, 15 for the Input Total and 15 for Valid Total. **Throughout the examples presented will be based upon Fitzsimmons, A. and Love, T., 'A Review and Evaluation of Software Science', ACM Computing Surveys, vol. 10, No 1, March 1978; duly expanded using the ISEB Foundation Certificate Syllabus for Software Testing.**

The internal logic of the function is depicted using the following low level notation which is 'white-box' in nature …

- Flow graph (i.e. graphical depiction of program logic)
- Program code listing (i.e. a Program Definition Language – PDL)
- McCabes' cyclomatic complexity metric
- Halsteads' size metric
- Inverted Decision Tree (i.e. for derivation of execution paths)
- Test Plan (i.e. for derivation of testing strategies from decision tree)
- Test Case Specification (i.e. derivation of test cases from test plan)
- Test Case Execution (i.e. physical user input and expected outcome)

Flow-Graph



The flow graph for the algorithm to compute the average of several numbers is displayed above and its corresponding program listing is shown below. Notice that the design includes compound conditions and loops.

```
00->   PROCEDURE   average
00->   REM         This procedure computes the average of 100
00->   REM         or fewer numbers that lie between bounding
00->   REM         values; it also computes the input count, the input total
00->   REM         and input average.
```

```
00->    PROCEDURE ACCEPTS value AS INTEGER ARRAY [1:100]
00->    PROCEDURE ACCEPTS maximum AS INTEGER
00->    PROCEDURE ACCEPTS minimum AS INTEGER
00->    PROCEDURE RETURNS average AS INTEGER
00->    PROCEDURE RETURNS total_input AS INTEGER
00->    PROCEDURE RETURNS total_valid AS INTEGER
01->    DEF i AS INTEGER
01->    DEF sum AS INTEGER
01->    LET i = 0
01->    LET sum = 0
02->    DO WHILE value[i] <> - 999
03->    AND total_input < 100
04->            LET total_input = total_input + 1
05->            IF value[i] >= minimum
06->            AND value[i] <= maximum
07  ->          THEN
07->                    LET total_valid = total_valid + 1
07->                    LET sum = sum + value[i]
08->            ELSE
08->                    REM – Logical Skip (i.e. perform no operation)
00->            END IF
09->            LET i = i + 1
00->    REPEAT DO
10->    IF total_valid > 0
11->    THEN
11->            LET average = sum ÷ total_valid
12->    ELSE
12->            LET average = -999
00->    END IF
13->    DISPLAY average
00->    END PROCEDURE
```

Please note that the prefix numbers (01 to 13) corresponds to each node on the flow graph. Also note that not all lines have a prefix (i.e. as designated by 00), as they are comment lines, additional control structure or procedure definition statements.

The McCabes' Cyclomatic complexity (or minimum execution paths) for the above program can be calculated by subtracting the count of all the nodes from the count for all the connections, plus 2. Thus subtracting 13 nodes from 18 connections and adding the value of 2 gives a value of 7. That is to say the above program has 7 execution paths within its logic.

Then Halstead's size metrics can then be applied thus.

- Operand 'average' occurs 3 times.
- Operand 'value' occurs 5 times.
- Operand 'total_input' occurs 4 times.
- Operand 'total_valid' occurs 5 times.
- Operand 'maximum' occurs 2 times.
- Operand 'minimum' occurs 2 times.
- Operand 'i' occurs 7 times.
- Operand 'sum' occurs 5 times.

There were 8 operands (n2), which occurred 33 times (N2) within the code. Thus an average 4.125 occurrences per operand.

Operator 'PROCEDURE' occurs 1 times.

- Operator 'REM' occurs 5 times.
- Operator 'PROCEDURE ACCEPTS' occurs 3 times.
- Operator 'PROCEDURE RETURNS' occurs 3 times.

- Operator ‘DEF’ occurs 2 times.
- Operator ‘WHILE’ occurs 1 times.
- Operator ‘AND’ occurs 1 times.
- Operator ‘DO WHILE’ occurs 1 times.
- Operator ‘LET’ occur 8 times.
- Operator ‘IF’ occurs 2 times.
- Operator ‘THEN’ occurs 2 times.
- Operator ‘ELSE’ occurs 2 times.
- Operator ‘END IF’ occurs 1 times.
- Operator ‘REPEAT DO’ occurs 1 times.
- Operator ‘END PROCEDURE’ occurs 1 times.
- Operator ‘ELSE’ occurs 1 times.
- Operator ‘DISPLAY’ occurs 1 times.

There were 17 operators (n1), which occurred 39 times (N1) within the code. Thus an average of 2.43 occurrences per operator.

Halstead showed that the length N could be estimated from the equation

$$N = n1 \log_2 n1 + n2 \log_2 n2$$

And program volume may be defined as

$$V = N \log_2 (n1 + n2)$$

It should be noted that V will vary with the programming language and represents the volume of information (in bits) required to specify a program.

Halstead also defined that there must be a theoretical minimum for the volume for any particular algorithm. The ratio L defines the volume of the algorithm in its most compact form.

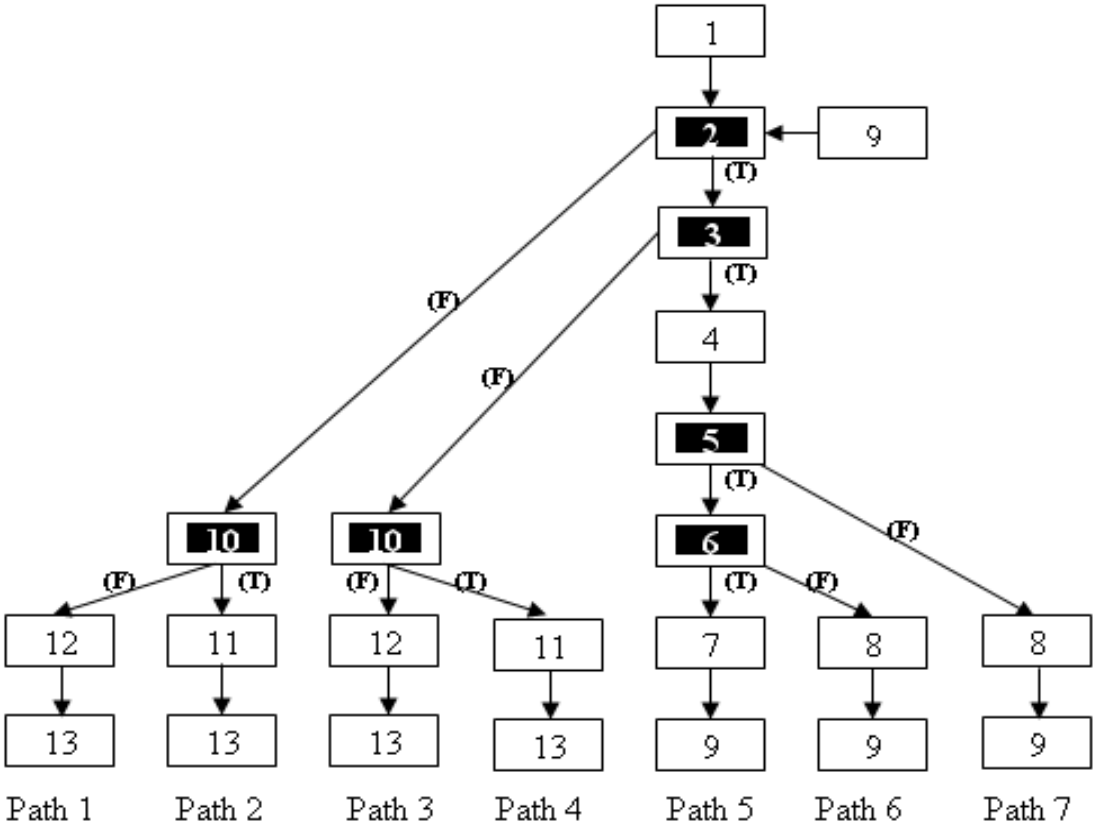
$$L = (2 \div n1) * (n2 \div N2)$$

Halstead further proposed that each language might be categorised by a language level with high level languages such as English Prose having a mean value of 2.16 and low level Assembler having a value of 0.88! However, others have argued that the language level should include a function for the programmer’s experience and ability along with other factors such as the development environment.

By analysing McCabes’ Cyclomatic complexity metrics and Halstead’s size metrics it is possible to gauge whether functions and procedures within the software would benefit from further levels of abstraction; i.e. achieved through refactoring of the code into smaller components. The concept being to maximise cohesiveness and minimise coupling through compact and efficient code that is defined and further refined in accordance with specified metric thresholds for size and complexity.

Once the SQA statistics have been computed the necessary white box testing can be conducted. First the base set of linearly independent paths through the flow graph are identified. We know the minimum number of execution paths from the Cyclomatic complexity value, which in our case was seven.

To greatly assist in identifying all the execution paths, it is advisable to redraw the flow graph as an inverted decision tree, as shown next, that is based on the Predicate nodes (i.e. conditional statements), where by the ‘trunk’ is represented by the commencement node and the ‘leaves’ represent the final execution node for that particular path.



Please note:

A node corresponds to a numbered circle on the Flow Graph, which in turn will correspond to the prefix numbers on the source code of the PDL. Hence the individual instruction statements for each node can be traced.

- Nodes 2, 3, 5, 6 and 10 are Predicate nodes (i.e. conditional loops and jumps).
- Node 1 is the 'trunk'. In other words the instigating node for the logic sequence.
- Nodes 13 and 9 are the final 'leaves' of the inverted decision tree.
- Node 13 is a terminator while node 9 feeds back into node 2 as part of a repeating conditional loop.

Thus from the above decision tree the following executions paths were derived …

*** TEST PLAN ***		
Execution Path	**Node Sequence**	**Testing Strategy**
1:	1-2-10-12-13	Initial Abort using (i.e. using –999)
2:	1-2-10-11-13	Abort Mid-Operation (i.e. using –999)
3:	1-2-3-10-12-13	Input more than 100 entries using all invalid values
4:	1-2-3-10-11-13	Input more than 100 entries using some / all valid values
5:	1-2-3-4-5-6-7-9-…	Input up to 100 entries using no invalid values
6:	1-2-3-4-5-6-8-9-…	Input up to 100 entries with some value(s) being invalid - i.e. they exceed the maximum value
7:	1-2-3-4-5-8-9-…	Input up to 100 entries with some value(s) being invalid – i.e. they reside below the minimum value.

Please note that ellipsis (…) following execution paths 5,6,7 indicate that any path through the remainder of the control structure is acceptable. It is likely that a decision tree may generate more

execution paths than the Cyclomatic complexity value, however, this is quite normal as often the branches of the decision tree repeat themselves and so ideally need to be 'pruned' to remove unnecessary replications. Hence it is normally the case that some sort of rationalisation of the various execution paths will be necessary (i.e. the ellipsis) for which the technique of 'Compartmentalisation' is recommended and this is discussed later within the section entitled 'Author's notes on Compartmentalisation'. Within this example the repeating execution paths (full or partial) are identified and then only one of these repeating execution paths is actually tested. The other repeating paths are assumed to be tested by virtue of them being substantially identical (in full or part) to the one tested.

Thus from the above executions the following test case specifications were drawn up:

*** TEST CASE SPECIFICATION ***	
TEST CASE:	PATH 1
Strategy:	Initial Abort (i.e. using –999)
Test Data:	Value (1) = - 999
Expected Results:	Average = -999; other totals at initial values.
TEST CASE:	PATH 2
Strategy:	Abort Mid-Operation (i.e. using –999)
Test Data:	Input up to 99 valid numbers terminating with a rouge value of –999.
Expected Results:	Correct average based on (n) values and proper totals.
Notes:	Cannot be tested stand-alone; must be tested as part of paths 5, 6 and 7.
TEST CASE:	PATH 3
Strategy:	Input more than 100 entries using all invalid values
Test Data:	Attempt to process 101 or more values. Of which first 100 values should be outside the valid range.
Expected Results:	Correct average based on (n) values and proper totals.
TEST CASE:	PATH 4
Strategy:	Input more than 100 entries using some / all valid values
Test Data:	Attempt to process 101 or more values.
Expected Results:	Correct average based on (n) values and proper totals.
TEST CASE:	PATH 5
Strategy:	Input up to 100 entries using no invalid values
Test Data:	Value(i) = valid input where i < 101
Expected Results:	Correct average based on (n) values and proper totals, however, after inputting the 100th value the program should then automatically finish.
TEST CASE:	PATH 6
Strategy:	Input up to 100 entries with some value(s) exceeding max
Test Data:	Value (i) = valid input above minimum value where i < 101 However, at least one input must exceed maximum value!
Expected Results:	Correct average based on (n) values and proper totals, however, those entries that were above the maximum value were ignored within the computation for the averages and totals.
TEST CASE:	PATH 7
Strategy:	Input up to 100 entries with some value(s) below min
Test Data:	Value (i) = valid input where i < 101 However, at least one input must fall below minimum value!
Expected Results:	Correct average based on (n) values and proper totals, however, those entries that were below the minimum value were ignored within the computation for the averages and totals.

Assumptions

(1) i = Loop counter corresponding to user input

(2) n = Number of valid inputs (i.e. total_valid from program code listing)

(3) That within the program code, the values are passed using an integer value array that ensures a valid integer numeric values have already been entered with no decimals or alpha-numeric characters.

Now that the test strategies and test cases have been defined, it is necessary to select the appropriate input and expected outcome for the test runs. In particular the results returned should be verified using a calculator or checked against a spreadsheet containing suitable test inputs and expected test results.

Below is the test case implementation for the above test case specification. Please note that a test run may correspond to more than one test case and that a test case may require more than one test run. This is particularly necessary when testing for equivalence partitioning and boundary value analysis, as well as, when a test run is applicable for use with several of the other test cases and so it can be 'shared' so that the design and development of the Test Runs can be 'optimised' for efficiency.

***** TEST CASE IMPLEMENTATION *****

Case Run	1	2	3	4	5	6	7
A	X						
B		X			X		
C		X			X		
D		X			X		
E					X		
F				X			
G		X					X
H		X				X	
I				X			
J			X				

Strategy for Test Cases:

1. Initial Abort
2. Mid-Operation Abort
3. Input more than 100 entries using all invalid values
4. Input more than 100 entries using some / all valid values
5. Input up to 100 entries using no invalid values
6. Input up to 100 entries with some value(s) exceeding maximum value
7. Input up to 100 entries with some value(s) below minimum value

Test Run (A to J)	Inputs	Expected Outcome			Remarks
		Count	Total	Average	
A	-999	0	-	-	Aborts on entering rogue value
B	1, -999	1	1	1	Aborts on entering rogue value
C	1, 2, 3, 4, 5, -999	5	15	3	Aborts on entering rogue value
D	1, 2, 3, 4, 5, ... ,99, -999	99	4950	50	Aborts on entering rogue value
E	1, 2, 3, 4, 5, ... 99, 100	100	5050	50.5	Automatically exits after 100th entry
F	1, 2, 3, 4, 5, ... 99, 100, 101	100	5050	50.5	Automatically exits after 100th entry, value 101 never gets the opportunity to be entered.
G	-1, -2, -3, -4, -5, -999	5	-	-	Aborts on entering rogue value
H	1000, 2000, 3000, -999	3	-	-	Aborts on entering rogue value
I	+1, -1, ... ,+50, -50, +51	50	2525	25.25	Automatically exits after 100th entry; value 101 never gets the opportunity to be entered.
J	-1000,-1001,-1002, -1003, ..-1101	100	-	-	Automatically exits after 100th entry; value 101 never gets the opportunity to be entered.

- **The three dots (i.e. …) represents the intervening numbers within sequence. For example 1, 2, 3 … , 8, 9 would represent the values 4, 5, 6 and 7.**
- **Minimum valid value is 0.**
- **Maximum valid value is 500.**
- **Expected Outcome results were validated using spreadsheet simulations.**

Author's notes on Compartmentalisation

Compartmentalisation is an important concept as it used to rationalise the execution paths within the flow-graph that represents the program logic. In other words it is used for pruning the excess execution paths produced by the inverted decision tree, which is in turn was originally derived from the flow-graph.

Compartmentalisation is an acquired skill that is very important to develop, as it can save much time and effort during testing by identifying which execution paths are independent and which are dependant. The reason being that execution paths that are independent need only be tested once and can be done so in isolation, whereas execution paths that are dependant need to be tested for all their various combinations.

The criteria for an execution path to be independent is that it must consist of a segmented section of the flow-graph that can then be isolated and it must have only one entry point and only one exit point and its internal logic must be fully encapsulated so that its content is not subject to external influences such as dependency upon the previous juncture points within the execution path.

For an example of a set of independent execution paths, assume a system consists of three menus, each of which have three options, then the combinations for the independent execution paths within this three-tier menu system would be 9 (i.e. combined 3 + 3 + 3)

1st level Menu options would be **A**, **B**, **C**

2nd level Menu options would be **D**, **E**, **F** (not dependant on 1st level menu options)

3rd level Menu options would be **G**, **H**, **I** (not dependant on 1st or 2nd level menu options)

The resultant combination list of nine independent execution paths for this three-tier menu system would be …

- option **A** (1st level menu)
- option **B** (1st level menu)
- option **C** (1st level menu)
- option **D** (2nd level menu)
- option **E** (2nd level menu)
- option **F** (2nd level menu)
- option **G** (3rd level menu)
- option **H** (3rd level menu)
- option **I** (3rd level menu)

Dependant execution paths unfortunately cannot be rationalised as their current juncture point within the execution path is dependent upon the previous juncture points within the execution path.

For an example of a set of dependant execution paths, assume a system consists of three menus, each of which have three options, then the combinations for the dependant execution paths within this three-tier menu system would be 27 (i.e. compounded 3 * 3 * 3)

1st level Menu – **A**, **B**, **C**

2nd level Menu – **D**, **E**, **F** (dependant on 1st level menu options)

3rd level Menu – **G**, **H**, **I** (dependant on both 1st and 2nd level menu options)

The resultant combination list of twenty-seven dependent execution paths for this three-tire menu system would be …

- option **A** (1st level menu) ➔ option **D** (2nd level menu) ➔ option **G** (3rd level menu)
- option **A** (1st level menu) ➔ option **D** (2nd level menu) ➔ option **H** (3rd level menu)
- option **A** (1st level menu) ➔ option **D** (2nd level menu) ➔ option **I** (3rd level menu)
- option **A** (1st level menu) ➔ option **E** (2nd level menu) ➔ option **G**(3rd level menu)
- option **A** (1st level menu) ➔ option **E** (2nd level menu) ➔ option **H (**3rd level menu)
- option **A** (1st level menu) ➔ option **E** (2nd level menu) ➔ option **I** (3rd level menu)
- option **A** (1st level menu) ➔ option **F** (2nd level menu) ➔ option **G** (3rd level menu)
- option **A** (1st level menu) ➔ option **F** (2nd level menu) ➔ option **H** (3rd level menu)
- option **A** (1st level menu) ➔ option **F** (2nd level menu) ➔ option **I** (3rd level menu)
- option **B** (1st level menu) ➔ option **D** (2nd level menu) ➔ option **G** (3rd level menu)
- option **B** (1st level menu) ➔ option **D** (2nd level menu) ➔ option **H** (3rd level menu)
- option **B** (1st level menu) ➔ option **D** (2nd level menu) ➔ option **I** (3rd level menu)
- option **B** (1st level menu) ➔ option **E** (2nd level menu) ➔ option **G** (3rd level menu)
- option **B** (1st level menu) ➔ option **E** (2nd level menu) ➔ option **F** (3rd level menu)
- option **B** (1st level menu) ➔ option **E** (2nd level menu) ➔ option **I** (3rd level menu)
- option **B** (1st level menu) ➔ option **F** (2nd level menu) ➔ option **G** (3rd level menu)
- option **B** (1st level menu) ➔ option **F** (2nd level menu) ➔ option **H** (3rd level menu)
- option **B** (1st level menu) ➔ option **F** (2nd level menu) ➔ option **I** (3rd level menu)
- option **C** (1st level menu) ➔ option **D** (2nd level menu) ➔ option **G** (3rd level menu)
- option **C** (1st level menu) ➔ option **D** (2nd level menu) ➔ option **H** (3rd level menu)
- option **C** (1st level menu) ➔ option **D** (2nd level menu) ➔ option **I** (3rd level menu)
- option **C** (1st level menu) ➔ option **E** (2nd level menu) ➔ option G (3rd level menu)
- option **C** (1st level menu) ➔ option **E**(2nd level menu) ➔ option **H** (3rd level menu)
- option **C** (1st level menu) ➔ option **E** (2nd level menu) ➔ option **I** (3rd level menu)
- option **C** (1st level menu) ➔ option **F** (2nd level menu) ➔ option **G** (3rd level menu)
- option **C** (1st level menu) ➔ option **F** (2nd level menu) ➔ option **H** (3rd level menu)
- option **C** (1st level menu) ➔ option **F**(2nd level menu) ➔ option **I** (3rd level menu)

As can be seen this escalation of the complexity of the execution paths will require additional time, effort and skills during the testing process which are all costly. Hence the importance of applying Compartmentalisation to remove duplication and redundancy where possible as part of a rationalisation process.

NB: The technique of Predicate Field Matrix was used to compile the above list of dependant execution paths; this is detailed in the section entitled ‘Author’s notes on Predicate Field Matrix (i.e. as used to construct test cases)’

Author's notes on Flow-Graph (Execution Path) Testing Techniques

It would be quite correct to point out that the current thinking on software testing (as also prescribed by the ISEB syllabus) recommends that testing is performed by comparing the operation of the program against its formalised specification of requirements and not against the code itself. The concept being that the program logic may in fact itself be faulty and so it's apparent correct operation would be a distortion of the actual situation and hence the validity of the test findings would have to be called into question. As a result, some might argue that low level testing using pseudo code, flow graphs and size / complexity metrics as depicted in the previous example would be inappropriate, however, these techniques have been tried and tested for many years and are proven in generating comprehensive test documentation and procedures.

Furthermore these traditional testing methods can be modified to accommodate Human Computer Interface (HCI) testing; and in particular exploratory testing of the User Interface is an ideal candidate, especially if access to the source code is not possible or is denied. For example the pseudo code / source code listing could be dropped and the flow graph used instead to 'map' the operation of the User Interface where each node corresponds to a screen object (i.e. fields & buttons, etc) into which data is entered, amended or clicked to activate a function. This allows for the various execution paths within the screen views to be determined in a systematic and comprehensive approach which can then be used to determine the appropriate test cases. Indeed this very concept is demonstrated within the following sections.

- Author's notes on Equivalence Partitioning and Boundary Value Analysis
- Author's notes on the Predicate Field Matrix (i.e. as used to construct test cases)

Author's notes on Equivalence Partitioning and Boundary Value Analysis

Techniques available for testing the user input include Equivalence Partitioning and Boundary Value Analysis.

Example Employed …

A conference management system requires the input of the number of delegates that attended a seminar. The valid range would be from 1 person through to the maximum of 500 persons (which is the capacity that the conference centre can officially hold). Although the implementation will be specific to the programming being employed, it will be assumed here that the number of delegates will be entered into a text box on the screen, where upon it will be validated by the system and then transcribed to a memory store of type unsigned numeric integer. Therefore the specified constraints of the user input would be an unsigned-numeric-integer value in the range of 1 to 500 (inclusive). Alternatively, depending on the nature of the programming employed, this could be represented as a signed-positive-numeric-integer value in the range of 1 to 500 (inclusive). This play on words may or may not be important depending on the nature of the programming employed. In our example we have assumed it does not.

Implementation of Boundary Value Analysis …

With Boundary Value Analysis (also known as Range Extremity Testing) the test cases are devised to check the input extremities of a range. Any inputted value on or within the edges of an input range are allowed, those that reside beyond are rejected as shown below.

*** TEST STRATEGY / IMPLEMENTATION ***		
Testing Case Strategy	**Input Data**	**Expected Outcome**
Lower boundary less 1	0	Rejected as invalid (i.e. outside of boundary)
Lower boundary limit	1	Accepted (i.e. within limits of boundary)
Lower boundary plus 1	2	Accepted (i.e. within limits of boundary)
Upper boundary less 1	499	Accepted (i.e. within limits of boundary)
Upper boundary limit	500	Accepted (i.e. within limits of boundary)
Upper boundary plus 1	501	Rejected as invalid (i.e. outside of boundary)

Hence only six test cases are required for the boundary value analysis of the number of delegates attended:

- Values in the range of 3 through to 498 are assumed to be valid (but deemed not to require testing) by virtue that they fall within the boundaries of the valid range being tested; i.e. lower boundary plus 1 and upper boundary less 1.
- Values below 0 are assumed to be invalid (but deemed to not require testing) by virtue that they fall below the lower boundary less 1.
- Values above 501 are assumed to be invalid (but deemed to not require testing) by virtue that they fall above the upper boundary plus 1.

Implementation of Equivalence Partitioning …

With Equivalence Partitioning the input domain of a program is sub-divided into data classes from which specific test cases can be derived. That is to say the various components of the input domain are tested according to their corresponding data types, lengths, decimal places and formatting, etc. In addition some inputs are required to be within a range, others must be of a specific value or a member of a set.

Generally speaking, within Boundary Value Analysis it is often clear how the input data should be determined, however, within Equivalence Partitioning the cognitive process can be rather more demanding.

One recommended approach is to identify all the potential characteristics of the input field (regardless of whether they are valid or invalid), and then determine which of these characteristics should be accepted within the validation and which should be rejected.

In the case of the number of delegates attended, the class characteristics of the input data would be an unsigned-numeric-integer value in the range of 1 to 500 (inclusive). Alternatively this can be represented as a signed-positive-numeric-integer value in the range of 1 to 500 (inclusive). This play on words may or may not be important depending on the nature of the programming employed. In our example we have assumed it does not.

The range validation has already been catered for by the Boundary Value Analysis and so would not be considered here, however, an arbitrary valid value of 99 will be used so that the input is not rejected for being outside of the valid range. Neither is the length considered here as the input value will be a numeric value and not an alpha-numeric or a date value, etc. However, it is assumed that it is possible for an alpha-numeric or a date value to be entered within the text box. Again, depending on the nature of the programming employed, the validation of the inputted numeric value may or may not consider the sign (+ / -) and decimal places. In our example we will assume that it does.

Taking all this information into account, we can start to devise a testing strategy based upon a block analysis of the Data Types as shown below.

*** TEST STRATEGY ***					
Block Analysis Break-Down of Data Type					**Test ID**
Data Type	**Null**				A
	Date				B
	Alpha Numeric				C
	Numeric	**Signed**	**Integer**	**Pos (+) Ve**	D
				Neg (-) Ve	E
			Decimal	**Pos (+) Ve**	F
				Neg (-) Ve	G
		Unsigned	**Integer**	**Pos (+) Ve**	H
				Neg (-) Ve Caveat!	I
			Decimal	**Pos (+) Ve**	J
				Neg (-) Ve Caveat!	K

Ideally the best starting point is to identify all those characteristics that will be accepted. This is referred to as ‘positive’ or ‘+ve’ testing where by only the valid test cases are being sought as shown below.

*** TEST IMPLEMENTATION – TEST ID: D& H ***						
Test ID	**Constituent Parts of the Data Type**				**Sample Input**	**Expected Outcome**
D	Numeric	Signed	Integer	Pos (+) Ve	+99	Accept
H	Numeric	Unsigned	Integer		99	Accept

Test cases D and H are exactly the same in value; however, they are being viewed from a slightly different perspective in terms of the sign employed. It can be argued that test case H is redundant, as a positive sign value could not be entered for an unsigned integer. That is to say, for unsigned numeric values the + / - keys would have no effect upon the input, however, in our example it is assumed that they can be entered and so test case H is included. Test case F will be Rejected because even though it is a positive sign value, it is also a decimal value and not an integer value (i.e. whole number).

Next, ‘negative’ or ‘- ve’ testing is undertaken where by the characteristics of the input value which will be rejected are identified. By analysing all the various possible alternative characteristics and combining them together in various permutations it is possible to identify suitable test cases that will be rejected as shown below.

*** TEST IMPLEMENTATION – TEST ID: E, G, F, I, K & J ***						
Test ID	**Constituent Parts of the Data Type**				**Sample Input**	**Expected Outcome**
E	Numeric	Signed	Integer	Neg (-) Ve	-99	Reject
G	Numeric	Signed	Decimal	Neg (-) Ve	-99.99	Reject
F	Numeric	Signed	Decimal	Pos (+) Ve	+99.99	Reject
I	Numeric	Unsigned	Integer	Neg (-) Ve	-99	Reject
K	Numeric	Unsigned	Decimal	Neg (-) Ve	-99.99	Reject
J	Numeric	Unsigned	Decimal	Pos (+) Ve	99.99	Reject

It can be argued that test cases I and K would be impossible to enter as you cannot enter a negative value within an unsigned numeric data-type. That is to say, for unsigned numeric values the + / - keys would have no effect upon the input, however, in our example it is assumed that they can be entered and so test cases I and K are included.

Finally, 'exception' testing can be employed to include other possible scenarios such as null or blank values, as well as alpha-numeric and date inputs within a numeric field as shown below.

*** TEST IMPLEMENTATION – TEST ID: C, B & A ***			
Test ID	**Constituent Parts of the Data Type**	**Sample Input**	**Expected Outcome**
C	Alpha Numeric	XXXX	Reject
B	Date	11/04/69	Reject
A	Null or Blank		Reject

Summary …

At this point the compilation of the test case for the number of delegates is complete. This process would then be repeated for each of the other input fields within the system.

One important aspect to consider when devising test scripts for input field validation, is that a field (or any other screen object) may or may not be displayed depending on certain specified criteria or requirements. That is to say, one or several input fields (or even underlying database fields and memory variables) must contain certain specified values in-order for the input field (or any other screen object) to be displayed / enabled. Thus if applicable, the 'displayed / enabled' criteria should also be included within the test scripts for input validation.

Author's notes on Predicate Field Matrix (i.e. as used to construct test cases)

Another useful technique for identifying test cases based on the user input is the Predicate Field Matrix that exercises the various combinations within the Predicate (Conditional) Fields. The Predicate fields themselves are dynamic in nature as they contain different data for each test run; whereas the non-Predicate fields are static in nature as they retain the same or similar data from one test run to another.

Indeed the technique of using a Predicate Field Matrix is a similar in nature to Boundary Value Analysis and Equivalence Partitioning techniques, however, in this case the possible variations are exercised.

For example, let's assume that a data entry for a Tennis and Gym Club Membership record may consist of the following subset of input fields …

- Member ID
- Member Name
- Member Address and Post Code
- Member Telephone
- Member Gender: either 'Male' or 'Female'.
- Member Section: either 'Junior' (i.e. under 16), 'Adult', or 'Senior Citizen'.
- Gym Member: either 'Yes' or 'No'.
- Tennis Member: either 'Yes' or 'No'.

Points to note …

- Member ID is a unique identification number.
- Member Name, Address, Post Code and Telephone are non-Predicate fields (i.e. static in nature). For testing purposes they can contain standardised inputs, for example with alpha-numeric text fields the descriptions can be used as the data inputs. Likewise, with date and numeric fields, sensible values should also be used so that they can be easily verified and validated by visual inspection.
- Member Gender, Member Section, Gym Member and Tennis Member are Predicate fields (i.e. dynamic in nature). From these Predicate fields the various combinations of values can be determined from which the corresponding test cases can be derived.
- It is assumed that Social Membership is automatically included and so will not need recording on the membership record. This means that a member may still be classed as a Social Member even though they do not have Tennis or Gym membership. Hence the Social Member field would be classed as a non-Predicate field as it is both static and consistent with the same value for all records.

From the Predicate field permutations of Member Gender, Member Section, Gym Member and Tennis Member the following twenty-four test cases were derived by exercising each possible combination …

*** Predicate Field Matrix ***				
Case:	**Member Gender:**	**Member Section:**	**Gym Member:**	**Tennis Member:**
1	Male	Junior	Yes	Yes
2	Male	Junior	Yes	No
3	Male	Junior	No	Yes
4	Male	Junior	No	No
5	Male	Adult	Yes	Yes
6	Male	Adult	Yes	No
7	Male	Adult	No	Yes
8	Male	Adult	No	No
9	Male	Senior Citizen	Yes	Yes
10	Male	Senior Citizen	Yes	No
11	Male	Senior Citizen	No	Yes
12	Male	Senior Citizen	No	No
13	Female	Junior	Yes	Yes
14	Female	Junior	Yes	No
15	Female	Junior	No	Yes
16	Female	Junior	No	No
17	Female	Adult	Yes	Yes
18	Female	Adult	Yes	No
19	Female	Adult	No	Yes
20	Female	Adult	No	No
21	Female	Senior Citizen	Yes	Yes
22	Female	Senior Citizen	Yes	No
23	Female	Senior Citizen	No	Yes
24	Female	Senior Citizen	No	No

The estimation of the number of test cases produced can easily be calculated by multiplying together the number of possible values for each Predicate field. Again using the Tennis and Gym Club Membership record example …

- Member Gender has 2 values - Male or Female.
- Member Section has 3 values - Junior (under 16), Adult, or Senior Citizen.
- Gym Member has 2 values - Yes or No.
- Tennis Member has 2 values - Yes or No.

Hence 2 x 3 x 2 x 2 would result in 24 test cases as listed above.

In summary, although the Predicate field matrix technique is very efficient and effective in generating test cases, it can often produce too many test cases and so a process of rationalisation is required to manage the level of resultant testing; hence the need for compartmentalisation where possible.

Author's notes on the Documentation of Test Results

No formalised format for the documentation of Test Method and Results has been formally prescribed here. The reason being that it was the author's belief that these should be at the discretion of the QA Manager, however, as a general guide the following generic Test Script / Test Results sheet has been presented below.

*** GENERIC TEST SCRIPT / TEST RESULTS ***						
Program:				**Version:**		
Test Preparation / Set-Up and Test Execution Method Followed: NB: For a more rounded approach to testing, it is strongly advised that the test execution method is also documented as part of the test script and it should be cross-referenced to the test conditions listed below.						
Test ID	**Test Item**	**Test Condition/ Action**	**Expected Results**	**Actual Results**	**Pass/ Fail**	**Remarks:**
Remarks:						
Date:		**Tester Name:**		**Signature:**		

In truth the above generic test script / test results shown above is far too simplistic and stylised, however, it can be significantly refined for practical use.

For example, the inclusion of SQL scripts can be used to extract the data assignments made by the program using a dissimilar but functionally compliant mechanism that is independent of the program under test. This can provide evidence for validation and verification purposes.

HINT: The construction of the test scripts could be commenced by implementing the following technique. For each program, identify every system feature within its specification, and then for each feature identify every data field, runtime user input and function employed; identifying every possible value (as appropriate). From this definition of the system and its boundaries you can then construct a test case to exercise every combination of program, function, field and corresponding input value covering both 'positive' (+ve) testing and 'negative' (-ve) testing.

The following is an example of a Generic Test Script …

Generic Test Script / Test Results

No.	Test	Test Condition	Expected Results	Actual Results	Results	Comments
1	Feature to be tested: R990 FSA Loan Book Movement Report - Main Section – 1 &3 .2 on pages 2 & 3					
11	FSAR052	All MLAR records where ACCT320 MLAR Loan Category = 'O'	FSAR052 = Sum of MLAR027 Capital Repaid / 1,000	FSAR052 on report differs to that on DB	Fail - Will require programmer investigation & resolution	select (sum(mlar027)/1000) from mlar, acct where mlar001 = acct001 and mlar008 = COMP260 and mlar005= 'MLAR005' and acct320 = 'O'
x	Feature to be tested? i.e. Source of Reference					
nn	Describe test here	Condition 1	What's expected?	What's the actual results?	Pass or Fail	Any applicable comment - this may be a reference to the support sheet where the user can attach things like screenshots etc NB: / represents division
		Condition 2	What's expected?	What's the actual results?	Pass or Fail?	

NB: Appropriate value for COMP260 and MLAR005 are substitutes as necessary.

Presenting the test script in an A4 Landscape - Matrix Style format is OK for basic testing, however, it can be extremely cumbersome and restrictive in its practical use due to the size constrictions of the paper dimensions. Alternatively the test scripts can be presented in an A4 Portrait 'Memo' Style format. This format is more convenient and amenable for use as it can be readily expanded as shown below.

A populated version corresponding to the first test item …

*** Generic Test Script / Test Results ***	
No.	11
Feature to be tested?	R990 FSA Loan Book Movement Report - Main Section – 1 &3 .2 on pages 2 & 3
Test?	FSAR052 - Other Sec: Repayment of Principal
Test Condition?	All MLAR records where ACCT320 MLAR Loan Category = 'O'
Expected Results?	FSAR052 = Sum of MLAR027 Capital Repaid / 1,000
Actual results?	FSAR052 on report differs to that on DB
Result?	Fail - Will require programmer investigation & resolution
Remarks / Comments	select (sum(mlar027)/1000) from mlar, acct where mlar001 = acct001 and mlar008 = COMP260 and mlar005= 'MLAR005' and acct320 = 'O' NB: / represents division (÷)

A generic version is depicted on the next page …

*** Generic Test Script / Test Results ***	
No.	Nn
Feature to be tested?	i.e. Source of Reference
Test?	i.e. DB or View Field
Test Condition?	i.e. Filter Criteria
Expected Results?	i.e. Field Assignment
Actual results?	i.e. Outcome
Result?	i.e. Pass / Fail
Remarks / Comments	i.e. Any applicable comment - this may be a reference to the support sheet where the user can attach things like screenshots or SQL, etc

The other option, which is the best of both worlds, being that the A4 Landscape 'Matrix' Style format can be used as a top-sheet summary whilst the A4 Portrait 'Memo' Style format can be used for the detail.

Another issue relates to conditional logic where by the resultant outcome can be either True or False (Boolean operation), however, the above test scripts only cater for one possible outcome (i.e. True). In this scenario two test cases would be required, the first to handle the True condition, the other to handle the NOT True (or False condition). True or False outcome would be exercised through Positive and Negative testing as discussed in the earlier section entitled "Author's notes on Equivalence Partitioning and Boundary Value Analysis".

Finally, IEEE 829 does include a detailed specification for the recording of test result as detailed below; however, the above normally suffices in most cases.

IEEE 829 – 1998 Standard for Software Test Structure

Test log ...

- 9.2.1 Test log identifier;
- 9.2.2 Description;
- 9.2.3 Activity and event entries - Execution description / Procedure results / Environmental information / Anomalous events / Incident report identifiers.

Test incident report ...

- 10.2.1 Test incident report identifier;
- 10.2.2 Summary: Summarise the incident;
- 10.2.3 Incident description - Inputs / Expected results / Actual results / Anomalies / Date and time / Procedure step / Environment / Attempts to repeat;
- Testers / Observers;
- 10.2.4 Impact.

Test summary report ...

- 11.2.2 Summary;
- 11.2.3 Variances;
- 11.2.4 Comprehensiveness assessment;
- 11.2.5 Summary of results;
- 11.2.6 Evaluation;
- 11.2.7 Summary of activities;
- 11.2.8 Approvals.

Briefing on employing Multi-User Testing

Introduction and background …

There is no definitive approach to multi-user implementation; however, the following has successfully been tried and tested for many years by the author and is typical of many commercial systems:

1. Employ shared database access with individual record locking using pessimistic strategies that immediately lock a record upon commencing of screen based data input, editing or deletion. The problem with optimistic strategies are that they frequently cause issues for the users; namely they are expected to undertake the difficult task of identifying and choosing which of the conflicting data items are to be retained and which are be discarded during the save operation. Alternatively the users have to abort a data entry session and in doing so waste their time and effort. Likewise the problem with employing no record locking strategies is that several users can use the system simultaneously and literally overwrite each other's data without restriction; this can lead to serious issues with data omissions, duplications and discrepancies.

2. Employ a proven approach of read a recordset, write a record and any transaction processing are initially stored in temporary files to await future posting. The temporary files would consist of 'uncommitted account postings' and also functional processing 'to-do' lists. All data processing is conducted within a secure and encapsulated SQL environment; with exception handling and roll-back operation provided for when issues arise. Where such an approach cannot be implemented, particularly with complex or critical data processing operations (e.g. financial transaction postings), then mandatory single user mode of operation would be enforced using implied locking protocols embedded within the program logic. Implemented either system wide or per function as appropriate; with each function having its own locking schema (i.e. encapsulate subset of tables).

3. Employ counts and totals that are calculated on-demand rather than stored on file as these can become out of sync; this would be due to the differential caused by delays in database update operations and the functional implementation of the system design. If counts and totals are to be retained on file then they should be 'time-stamped' accordingly so that users know that they are looking at historic and not immediate data.

4. Employ data processing functions that have been rationalised to prevent record locking issues and dead-lock conflicts; these can arise when several users are accessing the same set of files and records concurrently. The intention is that each function is compartmentalised into discrete components that are executed in sequence to ensure that only the minimum of files and records are locked at any one time whilst data is securely inserted, deleted and updated. For example within a sales order entry (SOE) system an order would be converted into an invoice using several distinct and separate functions rather than a single function. The first being the 'order book-out' function within the SOE screen which would initially mark the order as ready for invoicing and assign any appropriate calculated values and flag settings. The invoicing of the order would then be conducted by two separate automated functions; the first moves flagged orders to historical storage and creates the necessary pro-forma transactions within the 'to-do' list. The second function generates the invoices accordingly from the 'to-do' list. Both of these invoicing functions can be run consecutively or separated by a substantial period of time and are designed to perform batch processing safely upon a collection of database records; particularly important if the data has to be written in a specific sequence or if there are critical fields to be updated. Batch processing functions are normally performed during dayend or overnight whilst no other user is accessing the system; hence avoiding potential concurrency issues.

5. Employ a timer based refresh facility for record lists and provide a manual button on record screens. The timers should be configurable and the user should always have the option to refresh whilst viewing records.

6. Employ a clear definition for the operation of the automated ID generation facility within a multi-user environment; such functions are necessary to prevent locking issues and duplicate ID allocations. For example the following protocols would be appropriate …

- For speed and convenience the highest ID's in each database table would be recorded separately within a locking table; where data is stored over several sub files (live / history / archive) then these would be checked accordingly during ID allocation to ensure that the highest ID is maintained throughout.

- Upon attempting to write the new assigned ID to a locking table the necessary system level record locks are engaged; where the ID is locked by another user who is assigning their own ID then a predetermined number of iterations of given length are used to provide a suitable delay before trying to relock the ID again. When the lock has been unsuccessful a message is displayed to the user explaining the situation and they are asked to wait a short while whilst the other user finishes; then when ready to recommence they then click the OK button onscreen to proceed. Upon doing so the whole ID locking procedure recommences.

- Upon attempting to write the ID to locking table a check is made to ensure that the ID has not already been taken. Where this has been taken then the next in sequence is chosen and the whole ID locking procedure recommences.

- Gaps in the ID sequence will occur as result of multi-user conflicts during the ID locking process; any mechanism used to reuse prior ID would need to be robust and effective within a multi-user environment to prevent duplicate ID's being issued. In most cases it is easier (and safer) not to try and to reuse prior ID's; instead a report should be generated identifying those ID's that were skipped.

7. Employ suitable Auto ID and User ID mechanisms. Auto ID represents automatic generated record identification and User ID' represents user assigned record identification. The operation of Auto ID's at database level is a viable option, particularly as many modern database systems can undertake the automated increment of primary key's themselves, however, the underlying technology and stated operation of the Auto ID's at database level must be thoroughly understood and trusted for the task. In many cases such Auto ID's mechanisms may not incorporate configurable prefixes or number formatting facilities. Likewise many Auto ID's mechanisms do not properly handle multi-user operation over a substantive network and nor can many handle the reuse of previously skipped records ID's. User ID's are less prohibitive in operation but like the Auto ID's these fields must be locked to prevent editing once they are assigned as they are intended to be unique values. Again a similar prefix and numeric formatting mechanism as used on the Auto ID's can be applied to the User ID's for generating default values that can be overwritten by the user before committal. Many argue that it is better to implement hard coded Auto ID and Users ID mechanisms within the software code (as briefly described in item 6 above) so that underlying operation is a known entity that can be duly inspected.

The multi-user mechanism employed by the software should be operationally robust and functional; if not already specified then the system operation must be mapped accordingly as this forms the basis of the test strategy compilation. The investigation and determination of the underlying Multi-User and Record-Locking mechanisms is very important as they can be implemented in a variety of different ways and indeed antiquity has provided classic examples that range from using system level procedure based error traps within the programme code (e.g. ON ERROR … GOTO … RESUME as used in VB6 and VBA) through to implied protocols (these are operational rules embedded within the application logic itself and are intended to be strictly adhered to within the program design). Indeed screens typically use implied protocols to securely store data additions and changes in transient memory prior to committing to disk, whereas direct write is seldom employed due to the risk of corruption when several users tried to write the same set of records simultaneously without the necessary preventative measures as discussed above. For example many of the early programmable databases in the 90's were intended for single user operation as they just simply overwritten the target data, however, it was later found that these could also successfully be implemented within multi-user networks by employing implied protocols. A typical example of an implied protocol is the mechanism of having a specifically assigned file directory (i.e. folder location) for each database table and for every 'locked' record a corresponding ASCII text file would be generated with the record ID as the file name. Thus implied protocols were successfully employed that used simple file find facilities to identify if a record was locked by virtue of the existence of its corresponding ASCII file. The same

approach was also employed to manage concurrent users and often these text files were populated with useful debugging details such as when the record was created or when the user logged on. When the implied record lock was released then the corresponding ASCII text file would be deleted from disk. Likewise when the user logged out their corresponding ASCII text file would also be deleted from disk. The only disadvantage of this approach at the time was that in those days file name sizes were limited to eight digits with a three letter extension. Thus only one hundred million records could be stored per database file (i.e. record ID's ranging from 00000000 through to 99999999). Fortunately this was more than sufficient for most systems of that era and indeed many modern systems too! These days a similar mechanism is often employed, however, instead of files an internal record locks table is included within the database and instead of a 'file find' function a 'record find' function is used. Other than that the same principles are applied for implied locking.

Even in today's technology it is true to say that record locking an entire record set is problematic to implement and usually requires extensive resources, however, this can also be rationalised quite by simply locking the root record within a hierarchical record structure so that the related records would be locked by implied assumption. Other examples of implied protocols include the use of 'totem' locks which essentially enforce single user operation within multi-user systems when the functions being executed are either business critical (requiring human supervision), or the functions being executed are susceptible to corruption if several users accessed the same set of records simultaneously (particularly if there is a specific sequence in which the data must be written back and totals updated).

Also traditionally favoured by large multi-user systems was to employ a separate data processing computer which was used to update the database files from a set of serialised transaction batch files. Processing was undertaken on a periodic basis (usually overnight or during regular down-times). These transaction batch files were generated by the various features of the system that could have full read access to all files but no write access was permitted to any of their records. The advantage of this approach being that the risk of multi-user conflicts was in theory mitigated (i.e. as only one computer actually writes to the database then multi-user record locking was technically no longer needed if the implied protocols were strictly implemented), however, data additions and changes were not immediate and so can only be valid given a particular past date-time reference or 'time-stamp'. Hence totals and other statistics were not normally displayed within the immediate screen display, instead a menu option or screen 'button' had to be selected in order to run a set of corresponding SQL extraction scripts at runtime to produce counts, totals and other statistics. Serious problems did arise with this approach when programmers did not adhere to the strict implied protocols and instead directly wrote data back to the files rather than to the serialised transaction batch files.

Multi-User testing in practice…

The crux to the successful testing of Multi-User operation relies upon determining which system functions (and their underlying database tables) affects which of the other functions (and their underlying data tables) and in what specific order the underlying database tables are updated. Also necessary is to compile the menu and screen dialogue navigation for each function so that the testing can be duly specified and subsequently repeated.

Indeed a screen and its various component objects (i.e. fields, selection lists & function buttons, etc) would also be considered as separate functions. There also implied functions to be considered, in that within standard User Interface operation, it is normal to consider that entry access to a record on screen may be blocked or made automatic read-only if the corresponding record is currently displayed or edited by another user (and so locked). Likewise is it necessary to ensure that items on a selection list may still be selected even if their corresponding record is currently being displayed or edited by another user.

Hence the starting point being that each screen and its functions are tested against itself and against any screens that provide selection list content. Likewise the screen and its functions would also be tested against other screens and their functions that might potentially conflict; for example the immediate updating of counts, totals and other statistics.

Using the example of a Sales Order Entry (SOE) screen for data input and amendment, a minimum of two user workstations would be required to evaluate the interactions of each identified function within

the SOE screen against itself and all the other associated functions and screens that might be impacted by its operation. In addition several user workstations will be required to ensure selection lists can be accessed while their records are being locked for editing or viewing. For demonstration purposes, a nine user network will be considered here:

- Workstation #1 - View on screen (and so lock) a Product record.
- Workstation #2 - View on screen (and so lock) a Product Explosion record (i.e. SOE template).
- Workstation #3 - View on screen (and so lock) a Customer record including Sales Statistics.
- Workstation #4 - View on screen (and so lock) a Delivery record.
- Workstation #5 - View on screen (and so lock) a Department record.
- Workstation #6 - View on screen (and so lock) a Staff record.
- Workstation #7 - View on screen (and so lock) a Special Price record.

NB: In the SOE example, the screen content includes selection lists for Product record, Product Explosion record, Customer record, Delivery record, Department record, Staff record and Special Price record. Furthermore orders can either be 'Live' where data can be entered and amended or 'History' where data becomes read-only after being duly processed. Product Explosion record being a template that allows several product items to be added to the SOE order using only a single selection; thus saving the user time and effort during data input. In addition to selection lists there may (or may not) also be files whose records are updated by the system even though their data may not be displayed on screen; typically this would be files such as Sales Statistics that retain totals for the previous twelve months and associated summaries which are used for trends analysis purposes.

Within Workstations #1 to #7 the records displayed for Product, Product Explosion, Customer, Delivery, Department, Staff and Special Price should all be inter-related so that the record-locking for the Book-out function within the SOE screen can be tested accordingly. Furthermore a Special Price between the Customer and Product should be established and included within the Product Explosion.

- Workstation #8 – Used to evaluate SOE order screen (i.e. establish record lock)
- Workstation #9 – Used to evaluate SOE order screen (i.e. confirm record lock)

Workstations #8 and #9 are used to manually evaluate the record locking operation of the designated SOE screen and its CRUDE-FX functions. The matrix is formed by simultaneously pitting each function against itself and each other. This table assumes interaction with other associated screens and their locked records (i.e. Workstation #1 through to Workstation #7).

SOE - Two User - M/U Test Cases							
	C	**R**	**U**	**D**	**E**	**F**	**X**
C	1	2	3	4	5	6	7
R	8	9	10	11	12	13	14
U	15	16	17	18	19	20	21
D	22	23	24	25	26	27	28
E	29	30	31	32	33	34	35
F	36	37	38	39	40	41	42
X	43	44	45	46	47	48	49

Legend: CRUDE-FX functions

C - Create (i.e. New Record) * / ****
R - Read (i.e. Display Record or Print Record in R/O mode) ***
U - Update (i.e. Save Record) **
D - Delete (i.e. Delete Record) **
E - Edit (i.e. Amend Record) **
F - Function: Filter / Find (i.e. Extract and Sort Records)**
X - Exit Screen ***

NB:

* Represents that access to a record is only permitted by a single user at any one time. Where record-locking conflicts occur then the unsuccessful user(s) who were trying to access the

same record is informed that it has been already been locked by another and they are asked to try again after the other user (or processing task) has finished. This scenario occurs upon attempted entry in to the SOE screen from the Order List (i.e. record selection) screen.

.

** Represents that the function has the ability to detect network conflicts and can automatically resolve them by informing the user and asking them to try again after the other user (or processing task) has finished. This scenario occurs whilst the user is within the SOE screen.

*** Represents that one or several users may use this function simultaneously without record locking conflict or interference from each other or from system processing tasks.

**** Represents that the function should be repeated or varied between commencing from either the Order List screen or SOE screen.

R/O Read only mode. Where R/O is not stated then R/W (Read & Write) mode is implied.

In addition, in the scenario where several records can be selected from the Order List screen and edited collectively within the SOE screen, then it is recommended that several records should be selected and accessed through the primary test screen (workstation #8), however, on other workstations various orders within this selection should then be 'doctored' by means of insertion, deletion and amendment. On the primary test screen (workstation #8) these 'doctored' records should then be visited in sequence so that the operational stability and functional behaviour of the SOE screen can be verified and validated. This would be done using various permutations and combinations of actions that would normally be expected to raise an error if there were no error handling and record locking conflict recovery in place. That is to say the multi-select editing facility should be able to handle changes made by other users to the same set of selected records.

Finally to complete the multi-user checks on the SOE screen the following concurrent foreign key selections should be evaluated as additional test cases …

50. The SOE screen should simultaneously allow two or more users to enter the same PRODUCT within different orders.

51. The SOE screen should simultaneously allow two or more users to enter the same PRODUCT EXPLOSION within different orders.

52. The SOE screen should simultaneously allow two or more users to enter the same CUSTOMER within different orders.

53. The SOE screen should simultaneously allow two or more users to enter the same DELIVERY within different orders.

54. The SOE screen should simultaneously allow two or more users to enter the same DEPARTMENT within different orders.

55. The SOE screen should simultaneously allow two or more users to enter the same STAFF within different orders.

56. The SOE screen should simultaneously allow two or more users to enter the same SPECIAL PRICE for given Product / Customer combination.

NB: Essentially the selection lists should be read-only and not impacted by selections made by the other users.

Thus using the CRUDE-FX approach, there would be minimum of 49 test cases (7 functions x 7 functions = 49 test cases) required for each *independent* screen; plus a test case for each of the foreign key selection fields. In total 56 test cases were employed from the multi-user testing of the Sales Order Entry (SOE) screen for data input and amendment.

Briefing on CRUDE-FX and associated database implications

CRUDE-FX is an acronym for Create, Read, Update, Delete, Edit, Find / Filter and Exit. These form the implementation of the data input and file maintenance screen dialogues along with their underlying multi-user database operations and record locking; the implications of which are far reaching in terms of technical practicalities and constraints. This is particularly the case regardless of the database type employed; including flat-file, relational or object oriented.

One of the most significant issues to be considered is the cascading effects of CRUDE-FX operations upon the data and other users; particularly regards Update and Deletion. Most of the impact can be successfully mitigated through appropriate Multi-User and Record Locking operation, however, the knock-on effects of multiple record deletions and content changes can be potentially devastating. For example, the inadvertent deletion of a single job or invoice for a client would have minimal impact, however, if inadvertent cascading deletion of a client is permitted without constraint then it would be possible to delete all associated jobs, invoices and account transactions, etc for that client. This is compounded further if a specific client category had been inadvertently deleted then all associated clients and their jobs, invoices and account transactions would also be deleted. The escalating consequences of such actions are potentially devastating for all concerned including the software vendor; as often the resulting outcome can include litigation proceedings and adverse publicity.

A historically proven solution is simply to implement implied protocols that state only ‘unprocessed’ data can be deleted and that only one record may deleted at a time with the constraint that there must be no subordinate related records associated to it.

In addition, particularly for ‘processed’ records, the embedding of ‘deleted’ flags within records is encouraged so that essentially the system ignores any records that are flagged as ‘deleted’, however, if necessary it can be later manually unflagged by a system administrator and the database statistics and analysis automatically regenerated to reflect their resumed inclusion. Although this option requires additional complexity within the programming and database structure, it does mean that inadvertent deletions can be recovered with relative ease.

Another complementary approach being to move ‘processed’ records into separate files that the users can never amend or delete; such files would have designations such as ‘history’ and ‘archive’. The difference being that ‘history’ files are still used for analysis and reporting purposes, whereas, ‘archive’ is only used for historical enquiries and to keep the size of the ‘history’ files to a minimum. Although this option requires additional complexity within the programming and database structure, there is significant benefit in that as files are smaller and this means the processing is faster as there are fewer records to iterate through.

Finally, if the resulting database volumes are extremely excessive and seriously impacting upon performance then there is often no choice other than to trim the size of the database through PURGING (i.e. the permanent removal of records from a database). Typical candidate examples being Archived Jobs and Archived Invoices which are no longer used by the system and have been duly side-lined in to an subsidiary repository within the database. During the PURGING process it will be necessary to update the base-line figures for the Financials and other statistical reports so that although the purged records are removed their impact still remains.

Briefing on employing User Interface Testing

The test approach recommended for the User Interface (UI) is to establish a standard Test Matrix in which each of the UI Objects are subjected to specific Test Actions as determined by their associated Object Types.

Stage 1 - Compile a list of UI Objects …

- The ideal starting point for compilation of the UI Objects is the content of the system manuals. Alternatively UI Objects can be determined from conducting Flow Graph analysis of the various screen execution paths and their Predicate nodes; however, this is only necessary where suitable user documentation is not available to work from.

- Against each UI Object a list of the menu and dialogue navigation actions used to transverse the system would be included as an 'aide memoir'.

Stage 2 - Compile a list of Object Types …

- The various Object Types employed by the software for its user interface are then identified; these would include input / edit fields, function click buttons, tick boxes, radio buttons, menu selection field lookups (either fixed list or other database table lookup), etc. These would need to be categorised and listed accordingly before inclusion within the test matrix.

- Against each Object Type a set of standard attributes that need evaluating would be assigned; such as field length, data type, tab order, input and display formatting (including decimal places for numeric values) and valid range (if applicable), etc.

Stage 3 - Compile a list of Test Actions for associated Object Types …

- Using BS7925 as the terms of reference, the associated testing to be conducted is specified against each Object Type and its associated attributes; for example with numeric inputs both equivalence partitioning and boundary value analysis would be appropriate.

- Test matrix for the UI Object vs. Test Action would be duly compiled using the Object Type as the appropriate bind; thus providing a check list which can then be quickly and easily followed. This provides a systematic, convenient and repeatable approach to testing of the UI that clearly and concisely depicts the testing strategy employed.

The practical application of the above is clearly demonstrated using the following case study sampled from the content pages of the user / technical manual for a Sales Order Entry (SOE) system. The content pages provides an excellent framework and starting point for compilation of the UI Check List; indeed it can literally be used to identify the various menu options, system functions and specific screen objects to be tested.

For example, from a typical SOE application, the following excerpt was sourced from the user / technical manual …

Contents

The 'Contents' section can be used to compile a Check List (also known a Test Catalogue) providing the ideal vehicle for organising and focusing the UI testing process into a logical and manageable sequence that can be readily maintained and expanded. Indeed Test Catalogues are discussed in more detail in the following section entitled 'Briefing on Test Catalogues (for use in User Interface and Lifecycle testing)'.

In our case study we will focus on the entries relating to the Customer Module ...

The following is the Check List for the Customer Module that was constructed from the above sample. It is constituted from the Item Reference, UI Object and Object Type. Each group of items that collectively form a screen dialogue are accompanied by the Navigation path which is included in the top banner.

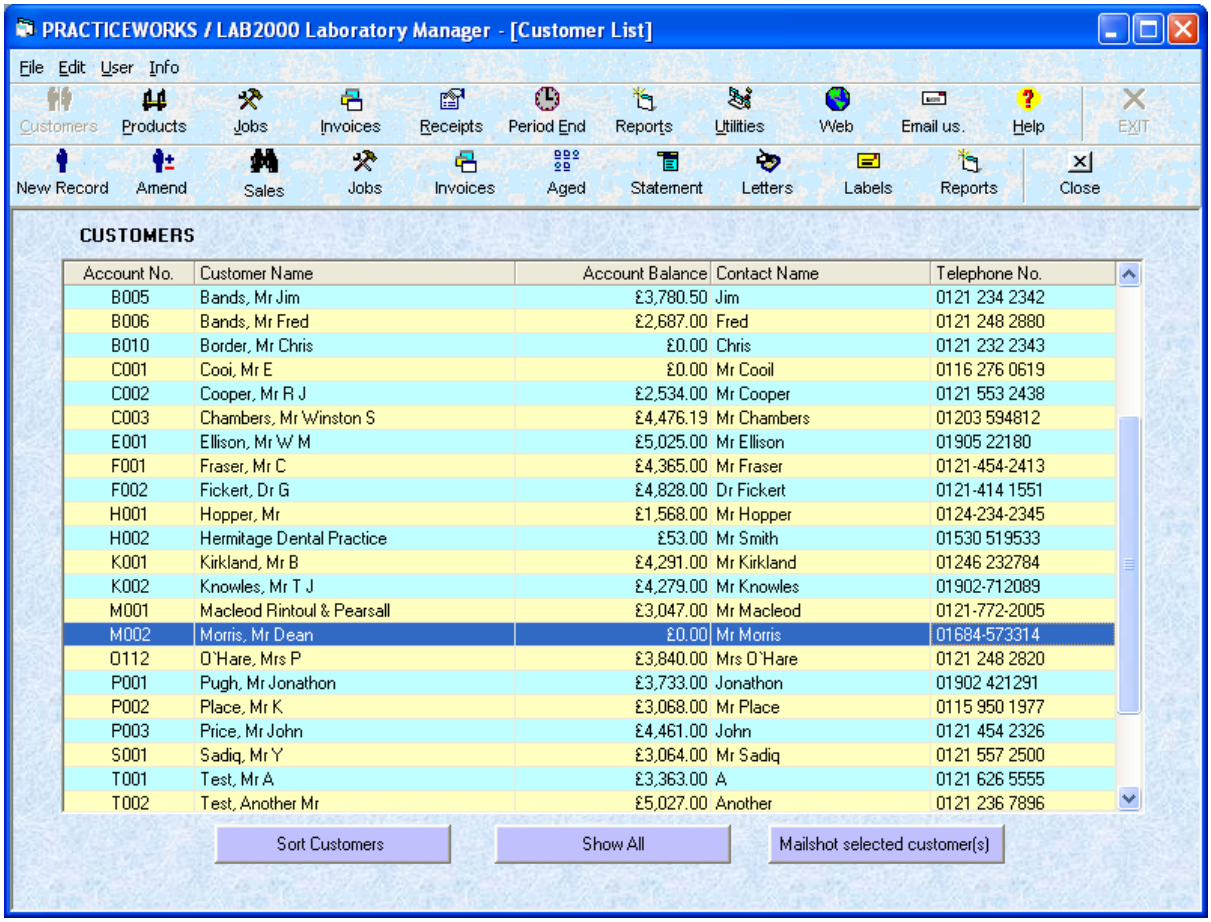
Navigation: 1. Main Screen > 4. Customers Module		
Item Ref.	**UI Object**	**Object Type**
4.	Customer module	Button (from SOE opening screen)
4.1.	Customer sub-module	Buttons and Records List
4.1.1.	Create customer	Form
4.1.2.	Amend customer	Form
4.1.3.	Deleting an existing customer record	Button
4.1.4.	Customer Functions	N/A (info only)
4.1.4.1.	Sales analysis	Button
4.1.4.2.	Customer details icons	N/A (info only)
4.1.4.3.	Customer jobs	Form
4.1.4.4.	Customer invoices	Form
4.1.4.5.	Customer aged balance	Form
4.1.4.6.	Customer statement	Form
4.1.4.7.	Customer letters	Form
4.1.4.8.	Customer labels	Form
4.1.4.9.	Customer reports	Form
4.1.5.	Close customer module	Button

Legend: Object Types

Form	Dialogue screen containing one or more of the below application objects.
Record List	Read Only record selection list presented in a matrix format.
Grid Column	Field Column within Record List.
Button	Function button activated by mouse click.
Input Field	Blank field for data entry.

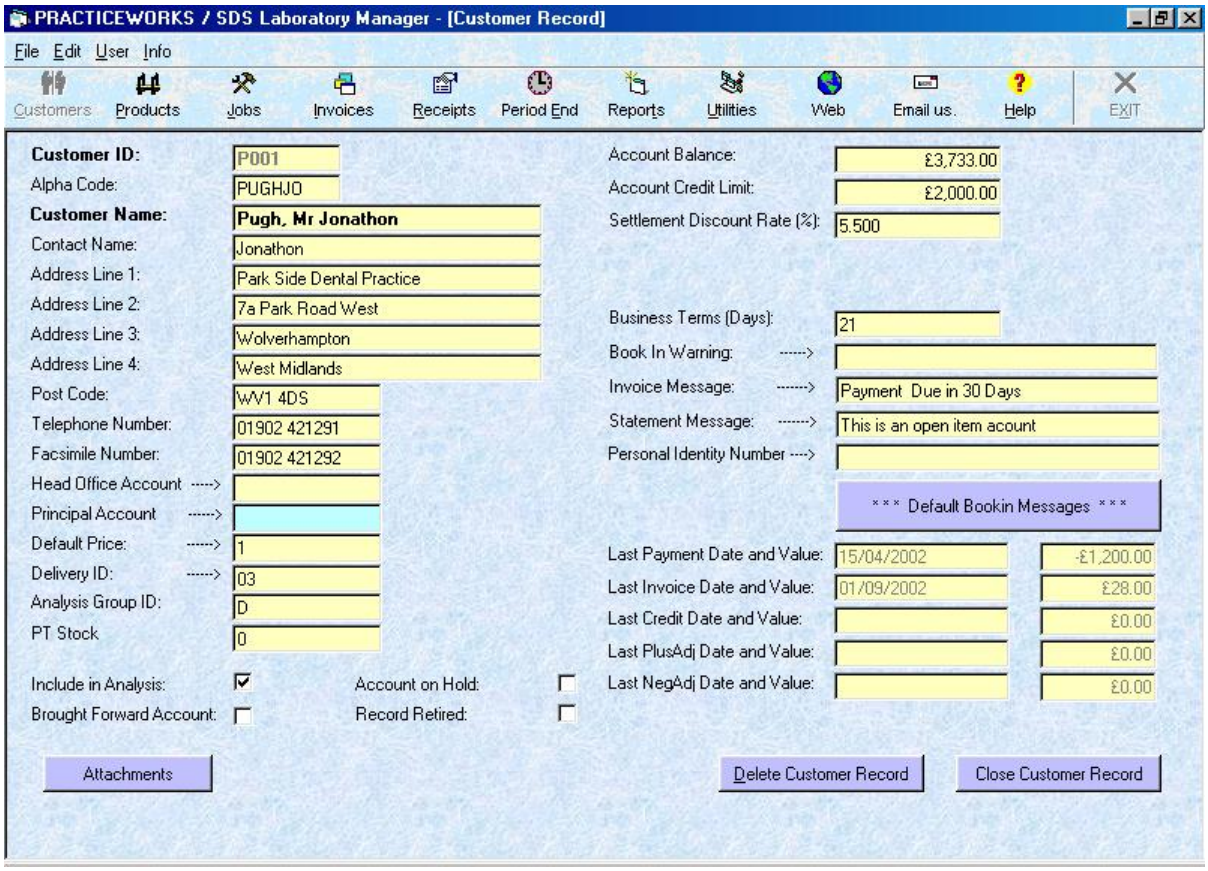
Input Field*	Blank field for data entry with a double-click facility to select from a list of foreign-key values; i.e. a pop-up selection list.
Amend Field	Field containing text for data amendment.
Amend Field*	Field containing text for data amendment with a double-click facility to select from a list of foreign-key values; i.e. a pop-up selection list.
R/O Field	Read Only field (cannot be modified by user).

Check List is then decomposed to further levels of detail and in particular to field level; for example item 4.1 – Customer Sub-Module - Icons and Records List could be broken down further using the corresponding screen dialogue for the Customer List as a guide …



Navigation: 1. Main Screen > 4. Customers Module (Customer List)			
Item Ref.	**UI Object**	**Object Type**	**Test Criteria**
4.1.	Customer module	Menu Icons and Records List	
4.1-(1)	A/c No	Grid Column	
4.1-(2)	Customer Name	Grid Column	
4.1-(3)	Account Balance	Grid Column	
4.1-(4)	Contact Name	Grid Column	
4.1-(5)	Telephone	Grid Column	
4.1-(6)	Filter / Sort	Button	Go-to Item: 28.4
4.1-(7)	Show All	Button	
4.1-(8)	Mail Shot Selected	Button	Test Script: E273

Likewise, item 4.1.2 – Amend Customer could be broken down further using the corresponding screen dialogue for the Customer Record as a guide …



Navigation: 1. Main Screen > 4. Customers Module (Customer List) > 4.1.2 Amend Cust			
Item Ref.	**UI Object**	**Object Type**	**Test Criteria**
4.1.2-(1)	Customer ID	Amend Field*	TXT len8
4.1.2-(2)	Alpha Code	Amend Field	TXT len10
4.1.2-(3)	Customer Name	Amend Field	TXT len50
4.1.2-(4)	Address Line 1	Amend Field	TXT len50
4.1.2-(5)	Address Line 2	Amend Field	TXT len50
4.1.2-(6)	Address Line 3	Amend Field	TXT len50
4.1.2-(7)	Address Line 4	Amend Field	TXT len50
4.1.2-(8)	Post Code	Amend Field	TXT len50
4.1.2-(9)	Fax Number	Amend Field	TXT len50
4.1.2-(10)	Head Office A/C	Amend Field*	TXT len8
4.1.2-(11)	Principle A/C	Amend Field*	TXT len8
4.1.2-(12)	Principle Name	R/O Field	TXT len50
4.1.2-(13)	Delivery Method	Amend Field*	TXT len2
4.1.2-(14)	Analysis Code	Amend Field	TXT len10
4.1.2-(15)	PT Stock Level	Amend Field	INT len10 rng0..99
4.1.2-(16)	Account Balance	R/O Field	-/+ CUR len12 dp2
4.1.2-(17)	Credit Limit	R/O Field	+ CUR len12 dp2
4.1.2-(18)	Settlement Discount	Amend Field	+ CUR len12 dp2
4.1.2-(19)	Business Terms	Amend Field	+INT len10
4.1.2-(20)	Bookin Message	Amend Field	TXT len8
4.1.2-(21)	Invoice Message	Amend Field	TXT len50
4.1.2-(22)	Statement Message	Amend Field	TXT len50
4.1.2-(23)	Hosp PIN Code	Amend Field	TXT len2
4.1.2-(24)	Last Payment Date	R/O Field	DTE
4.1.2-(25)	Last Payment Value	R/O Field	-/+ CUR len12 dp2
4.1.2-(26)	Last Invoice Date	R/O Field	DTE

Navigation: 1. Main Screen > 4. Customers Module (Customer List) > 4.1.2 Amend Cust			
Item Ref.	**UI Object**	**Object Type**	**Test Criteria**
4.1.2-(27)	Last Invoice Value	R/O Field	-/+ CUR len12 dp2
4.1.2-(28)	Last Credit Date	R/O Field	DTE
4.1.2-(29)	Last Credit Value	R/O Field	-/+ CUR len12 dp2
4.1.2-(30)	Last +Adj Date	R/O Field	DTE
4.1.2-(31)	Last +Adj Value	R/O Field	-/+ CUR len12 dp2
4.1.2-(32)	Last –Adj Date	R/O Field	DTE
4.1.2-(33)	Last –Adj Value	R/O Field	-/+ CUR len12 dp2
4.1.2-(34)	Include in Analysis	Check Box	T/F
4.1.2-(35)	B/F Account	Check Box	T/F
4.1.2-(36)	Account on Hold	Check Box	T/F
4.1.2-(37)	Record Retired	Check Box	T/F
4.1.2-(38)	Attachment	Button	
4.1.2-(39)	Clear Principle A/C	Button	
4.1.2-(40)	Clear Head Office A/C	Button	
4.1.2-(41)	Delete Customer	Button	
4.1.2-(42)	Default Book-in Messages	Button	

Legend: Test Criteria

NB: The Boundary Value Analysis parameters (i.e. valid values & date ranges) and the Equivalence Partitioning parameters (i.e. field length, type & formatting) are included with the UI Test Conditions column to provide a brief set of narrative instructions for use during testing; the decoding of which is as follows:

Field Type ...

TXT:	Text value; e.g. ABC
INT:	Integer value; e.g. 123
CUR:	Currency value; e.g. £456.78
DTE:	Date Format; e.g. 11/04/1969 (Y2K compliant format of DD/MM/YYYY).
T/F:	True/False value; e.g. True/False or Yes/No or Ticked/Unticked (Boolean).

Field Sign ...

-:	Negative numeric values permitted.
+:	Positive numeric values permitted.

Field Length ...

len__:	Length of field.

Decimal Places ...

dp__:	Decimal places of field.

Range …

rng_.._:	Range of valid values; e.g. 0°C to 100°C for liquid water

Other ..

Goto:	Go to test item
Test Script:	Perform specific functional test script (i.e. documented as per IEEE 829)

NB: Field tab sequence will be assumed to follow the order that the UI Screen Objects are listed within the Check List.

Essentially through a continuous process of incremental and cyclic refinement the Check List can be broken down to the level of detail that is deemed required. There is no maximum level of detail for the Check List, however, as it depends upon the circumstances at the time of compilation (which should also be documented for future reference).

Against each Object Type it is necessary to associate the corresponding Test Actions.

Object Type	Test Actions					
	A	B	C	D	E	F
Form (Overall operation & content)	X					
Record List	X					
Grid Column	X					
Button (function call)	X					
Input Field	X	X	X	X		X
Input Field *	X	X	X	X	X	X
Amend Field	X	X	X	X		X
Amend Field *	X	X	X	X	X	X
R/O Field	-	-	-	-		-
Check Box	X					X

Legend: Test Actions

A. Try out each function / screen object as detailed within the User & Technical manual to verify and validate their operational compliance to system spec and / or manuals.

B. Boundary Value Analysis (Valid Values and Data Ranges as per BS7925-2).

C. Equivalence Partitioning (Field Length, Field Type and Data Formatting as per BS7925-2).

D. Read-Only, Read-Write and Restricted Access to field contents.

E. Foreign Key Lookup and Selection.

F. Correct data storage and on-screen retrieval (i.e. check Save&Reload of New Records, Save&Reload of Changed Records, Save&Reload Additionally Changed Records)

-. Expected that test should be rejected by system (i.e. negative –ve testing)

X. Expected that test should be permitted by system (i.e. positive +ve testing)

Applying the Object Type / Test Actions to the Customer List …

Navigation: 1. Main Screen > 4. Customers Module (Customer List)									
Item Ref.	UI Object	Object Type [Test Criteria]	A	B	C	D	E	F
4.1.	Customer module	Menu Icons and Records List	X					
4.1-(1)	A/c No	Grid Column	X					
4.1-(2)	Customer Name	Grid Column	X					
4.1-(3)	Account Balance	Grid Column	X					
4.1-(4)	Contact Name	Grid Column	X					
4.1-(5)	Telephone	Grid Column	X					
4.1-(6)	Filter/Sort	Button [Goto Item: 28.4]	X					
4.1-(7)	Show All	Button	X					
4.1-(8)	Mail Shot Selected	Button [Test Script: E273]	X					

Likewise, applying the Object Type / Test Actions to the Customer Record …

Navigation: 1. Main Screen > 4. Customers Module (Customer List) > 4.1.2 Amend Cust									
Item Ref.	UI Object	Object Type [Test Criteria]	A	B	C	D	E	F
4.1.2-(1)	Customer ID	Amend Field* [TXT len8]	X	X	X	X	X	X
4.1.2-(2)	Alpha Code	Amend Field [TXT len10]	X	X	X	X		X
4.1.2-(3)	Customer Name	Amend Field [TXT len50]	X	X	X	X		X
4.1.2-(4)	Address Line 1	Amend Field [TXT len50]	X	X	X	X		X

Navigation: 1. Main Screen > 4. Customers Module (Customer List) > 4.1.2 Amend Cust								
Item Ref.	**UI Object**	**Object Type [Test Criteria]**	**A**	**B**	**C**	**D**	**E**	**F**
4.1.2-(5)	Address Line 2	Amend Field [TXT len50]	X	X	X	X		X
4.1.2-(6)	Address Line 3	Amend Field [TXT len40]	X	X	X	X		X
4.1.2-(7)	Address Line 4	Amend Field [TXT len50]	X	X	X	X		X
4.1.2-(8)	Post Code	Amend Field [TXT len50]	X	X	X	X		X
4.1.2-(9)	Fax Number	Amend Field [TXT len50]	X	X	X	X		X
4.1.2-(10)	Head Office A/C	Amend Field* [TXT len8]	X	X	X	X	X	X
4.1.2-(11)	Principle A/C	Amend Field* [TXT len8]	X	X	X	X	X	X
4.1.2-(12)	Principle Name	R/O Field [TXT len50]	-	-	-	-		-
4.1.2-(13)	Delivery Method	Amend Field* [TXT len2]	X	X	X	X	X	X
4.1.2-(14)	Analysis Code	Amend Field [TXT len10]	X	X	X	X		X
4.1.2-(15)	PT Stock Level	Amend Field [-/+INT len10]	X	X	X	X		X
4.1.2-(16)	Account Balance	R/O Field [-/+ CUR len12 dp2]	-	-	-	-		-
4.1.2-(17)	Credit Limit	R/O Field [+ CUR len12 dp2]	-	-	-	-		-
4.1.2-(18)	Settlement Discount	Amend Field [+ CUR len12 dp2]	X	X	X	X		X
4.1.2-(19)	Business Terms	Amend Field [+INT len10]	X	X	X	X		X
4.1.2-(20)	Book-in Message	Amend Field [TXT len8]	X	X	X	X		X
4.1.2-(21)	Invoice Message	Amend Field [TXT len50]	X	X	X	X		X
4.1.2-(22)	Statement Message	Amend Field [TXT len50]	X	X	X	X		X
4.1.2-(23)	Hosp PIN Code	Amend Field [TXT len2]	X	X	X	X		X
4.1.2-(24)	Last Payment Date	R/O Field [DTE]	-	-	-	-		-
4.1.2-(25)	Last Payment Value	R/O Field [-/+ CUR len12 dp2]	-	-	-	-		-
4.1.2-(26)	Last Invoice Date	R/O Field [DTE]	-	-	-	-		-
4.1.2-(27)	Last Invoice Value	R/O Field [-/+ CUR len12 dp2]	-	-	-	-		-
4.1.2-(28)	Last Credit Date	R/O Field [DTE]	-	-	-	-		-
4.1.2-(29)	Last Credit Value	R/O Field [-/+ CUR len12 dp2]	-	-	-	-		-
4.1.2-(30)	Last +Adj Date	R/O Field [DTE]	-	-	-	-		-
4.1.2-(31)	Last +Adj Value	R/O Field [-/+ CUR len12 dp2]	-	-	-	-		-
4.1.2-(32)	Last –Adj Date	R/O Field [DTE]	-	-	-	-		-
4.1.2-(33)	Last –Adj Value	R/O Field [-/+ CUR len12 dp2]	-	-	-	-		-
4.1.2-(34)	Include in Analysis	Check Box [T/F]	X					X
4.1.2-(35)	B/F Account	Check Box [T/F]	X					X
4.1.2-(36)	Account on Hold	Check Box [T/F]	X					X
4.1.2-(37)	Record Retired	Check Box [T/F]	X					X
4.1.2-(38)	Attachment	Button	X					
4.1.2-(39)	Clear Principle A/C	Button	X					
4.1.2-(40)	Clear Head Office A/C	Button	X					
4.1.2-(41)	Delete Customer	Button	X					
4.1.2-(42)	Default Book-in Messages	Button	X					

NB: In the above Check Lists the Object Type and Test Criteria have been merged into a single column to allow space for the Test Actions to be included.

The sample Check Lists for UI testing demonstrates the approach prescribed; however, they can be duly expanded further …

- Where considered appropriate to do so, the test matrix can be extended to include the Boundary Value Analysis parameters in more detail; namely:
  - Valid Values
  - Data Ranges

- Likewise, where considered appropriate to do so, the test matrix can be duly extended to include the Equivalence Partitioning parameters in more detail; namely:
  - Field Length

- Field Type
- Data Formatting

- Where considered appropriate to do so, the test matrix can be duly extended to include a record of the progress made during testing (i.e. by signing off each item as they are tested and recording other pertinent information such as date, time taken, tester's signature, pass / fail and remarks, etc). Essentially evolving the Test Matrix into a Test Catalogue.

- It would be possible to establish a database or spreadsheet containing a data dictionary of the UI screen objects along with their Object Types, Test Actions, Test Criteria and Navigation, etc. This can be printed on demand and used for testing purposes as and when required.

Finally, always watch out for the unexpected scenarios that need inclusion; for example function buttons such as [F1] to [F12] along with [Ins] and [Del] which may have corresponding function buttons provided for screen/mouse operation, however, the keyboard counterpart operation may not have the same underlying logic!

Briefing on Test Catalogues (for use in User Interface and Lifecycle testing)

Building upon the earlier section on User Interface Testing, the compilation of Test Catalogues is an extremely useful technique for establishing appropriate test documentation with the minimum of documentation and administration involved.

In the case of User Interface testing a Test Matrix is compiled and the list of screen objects is used to compile the Test Catalogue. In the case of Lifecycle testing the technical documentation (or reverse engineered specifications) is compiled to form a pictorial 'story board' with suitable annotates and narrative scripts; these depict the end to end operation (or lifecycle) of the system that then be can followed thorough by the testing, sales, support and training staff. The list of screens within the Lifecycle is used to compile the Test Catalogue.

Against each item within the Test Catalogue a record of the tester's signature and/or initials, date of sign-off and version tested would be included.

Indeed such an approach encourages comprehensive testing with the minimum of test scripts involved; thus both User Interface and Lifecycle testing can be rationalised accordingly.

Below is a brief sample of a Test Catalogue sourced from Page 73 (Briefing on employing User Interface Testing) …

Item Ref.	UI Object	Object Type	Tester	Date	Ver'n
4.	Customer module	Button (from SOE)	CP	23/03/12	4.2.4.1
4.1.	Customer sub-module	Buttons and Records List	CP	23/03/12	4.2.4.1
4.1.1.	Create customer	Form	CP	23/03/12	4.2.4.1
4.1.2.	Amend customer	Form	CP	23/03/12	4.2.4.1
4.1.3.	Deleting an existing customer	Button	CP	23/03/12	4.2.4.1
4.1.4.	Customer Functions	N/A (info only)	CP	23/03/12	4.2.4.1
4.1.4.1.	Sales analysis	Button	CP	23/03/12	4.2.4.1
4.1.4.2.	Customer details icons	N/A (info only)	CP	23/03/12	4.2.4.1
4.1.4.3.	Customer jobs	Form	CP	24/03/12	4.2.4.2
4.1.4.4.	Customer invoices	Form	CP	24/03/12	4.2.4.2
4.1.4.5.	Customer aged balance	Form	CP	24/03/12	4.2.4.2
4.1.4.6.	Customer statement	Form	CP	24/03/12	4.2.4.2
4.1.4.7.	Customer letters	Form	CP	24/03/12	4.2.4.2
4.1.4.8.	Customer labels	Form	CP	24/03/12	4.2.4.2
4.1.4.9.	Customer reports	Form	CP	02/04/12	4.2.4.7
4.1.5.	Close customer module	Button	CP	02/04/12	4.2.4.7

SIGN-OFF: This document has been compiled to attest that the SOE product has successfully completed its User Interface (UI) & Lifecycle testing as detailed above.

Briefing on Non-Functional and Other Testing

The non-functional and other system testing would be constituted by the multi-user operation, load performance operation, website operation, database volume generation and database content.

Testing of Multi-User Operation …

- Implemented through both manual operation and automated testing tools that simulate network traffic and record locking.

- Significant focus is placed upon evaluating the underlying record locking mechanisms and ensuring their adherence to the specified operation as detailed within the technical documentation.

Testing of Load Performance Operation …

- Implemented through both benchmarking of previous versions and appropriate evaluations of timings and resource usage for various system functionality. Essentially this can be as simple as employing a stop watch and a PC resource monitor showing memory, network, disk and processor peak usage.

- Used in conjunction with data volume generation in order to stress test the system. Typical data volume generation facilities would include application specific random data generators and multi-instance record copy function (e.g. both are used for creating sample customers, sites, jobs, invoices and account transactions, etc). This would allow database volumes to be generated on-the-fly both quickly and easily for testing, training and sales demonstration purposes. Also multi-instance record copy facilities would be a useful software feature for the clients and their end-users as it could be used to automate much of their initial data set-up.

Testing of Website Operation …

- Implemented through evaluation of usability, content, hyperlinks and operational robustness.

- Employing automated tools for website stress testing and speed evaluations.

Testing of Data Volume Generation …

- Implemented through both manual data entry and automated testing tools to generate appropriate database content. This includes employing random data generators and multi-instance copy functions. Indeed database volumes can be readily generated for projected life spans using estimated record numbers for given periodicals; for example X number of customers, Y number of Jobs / Invoices and Z number of Account Transactions for range of given years from 1 through to the envisage term of system usage.

- Used in conjunction with load performance testing in order to stress test the system.

Testing of Database Content …

- Visual inspections of the database content is conducted by comparing the screen inputs and report printouts with that of the database tables and fields; ideally a database schema should be provided specifying which database tables and fields are updated by which system functions and also in what sequence.

NB: Where appropriate the testing of Multi-User / Load Performance / Website / Data Volume Generation operation may be commercially subcontracted to specialist service providers.

Additional Non-Functional Testing activities ...

- Evaluate product Usability and Functional Compliance with stated description of functionality and features (i.e. checklist sourced from documentation & marketing literature, etc).
- Evaluate product Security including user access control, privileges and passwords.
- Evaluate product Installation Procedures; covering new installs, updates, and upgrades.
- Evaluate product Stability/Robustness and Reliability; covering error detection & recovery mechanisms.
- Evaluate product Backup & Restore and Data Import/Export/Repair facilities; covering disaster recovery and data transfer.
- Evaluate product Accessibility and Availability; covering workstation locations (e.g. office or remote access) and hours of access available to users (e.g. 9 to 5 Monday to Friday, etc).
- Evaluate Standard/Peak Usage, Inactivity & Downtime Levels; covering permissible downtime limits and known off-line processing requirements such as overnight and month end, etc. Also includes designated system Failure Recovery times and procedures (both manual and system). Used to gauge transaction processing throughput and resource requirements.
- Evaluate product Compatibility; covering integration of hardware, peripherals, networks, operating systems and other third-party software products.
- Evaluate product Localization and Internationalization operation; covering available languages, date and currency formats, etc.
- Evaluate product System Requirements and Technical Documentation; covering intended usage, support, training, marketing, and installation/update/upgrade, etc
- Evaluate product Scalability and Migration; including upsizing and downsizing, as well as, appropriate data conversions and necessary data transformations.
- Evaluate product Operational Readiness; covering deployment prerequisites including the necessary lead-times and administration involved in the ‘rollout’ and ‘going live’.
- Evaluate system capacity and growth; covering projected database volumes over successive years within the known lifespan of the product. Includes consideration as to the number of sites, users per site and computing equipment to be used over the years.

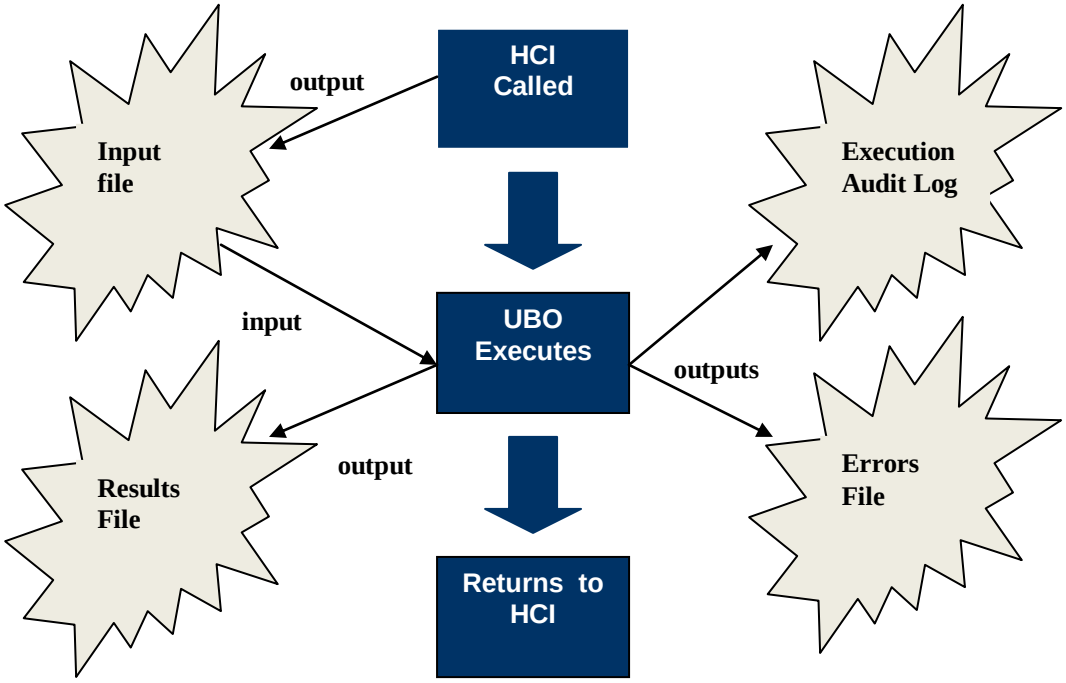
Other QA activities that support Non-Functional Testing ...

- Implement Quality Management System (QMS) with Metrics (for SLA adherence); ensuring that proper policies, procedures and documentation are in place with appropriate issue logging and monitoring facilitate.
- Implement regular QA Stakeholder meetings and Client/Staff consultation exercise.
- Implement QA Management model with particular emphasis on ‘Regression’ and ‘Acceptance’ testing to ensure that the product continues to remain as described, quality assured and fit for purpose.
- Compile library of ‘avoidance procedures’ and ‘workaround solutions’ for known issues.

Briefing on Semi-Automated & Inbuilt Testing Techniques

A well established and valuable technique for software testing concerns semi-automated input from the mouse and keyboard devices. Often it is perceived by many that these can only be simulated by employing expensive automated regression testing tools that can cost many thousands of pounds sterling (£), however, these can be cheaply and efficiently substituted by using a simple approach where by the 'inputs' are stored within transient input files (e.g. in ASCII text or XML format). These would be created automatically by the system operation but can also be easily created manually for testing purposes. Where by the software under development is designed or modified so that the program components are classified as being either a Human Computer Interface (HCI) or an Unattended Batch Operation (UBO). A HCI would be constituted by data views and menu navigation dialogues, where as a UBO would represent all other types of computer processing that does not require any user input (e.g. reports & analysis printouts, period-end routines, etc). The HCI components would contain their own validation (as appropriate to the application) and would contain the appropriate function buttons and menu selections to evoke the UBO's. When a UBO is evoked by a user action (e.g. such as a function button click or menu selection) it then calls a separate HCI screen that enters the run-time parameters for that UBO, alternatively the run-time parameters are taken directly from the instigating HCI, and these are then stored on a transient Input File that the corresponding UBO first reads upon starting. Also scheduled and overnight automated processing procedures may themselves directly call the UBO's. However, returning to our first example, the concept is depicted pictorially below.

Pictorial Representation of HCI and UBO Concept …



NB: The above schematic follows the standard universal function model

As an example let us assume that within an accounts system a user has selected a range of customer statements for re-printing. The HCI would be used to select on-screen those accounts for processing and it would also have an appropriate function button or menu selection that can be clicked to call the Statement Reprint facility. When the function button is clicked the HCI will first create the necessary Input file that would contain the list of customers (i.e. runtime parameters) and then it will call the corresponding UBO for the Statement Reprint facility. Upon commencement, the UBO will first read-in the list of customers from the Input File and then it will perform its task of reprinting the Statements for that list. The Customer Statements produced would be stored within the Results File for printing later. During processing the UBO will record any problems encountered with the Statement Reprint facility

within the Errors File and then upon completion it will return the program execution back to the calling HCI. In effect the HCI / UBO concept is similar in nature to a 'wrapper' (i.e. similar to parameter passing as used within program function and procedure calls). Furthermore it is a design feature that all outputted errors that are produced during the UBO are logged for post operation inspection within an Errors File; particularly useful for testing and fault finding. Likewise a record of the processing is retained within the Execution Audit Log and subsequently time-stamped for future inspection.

The concept is essentially to allow the programming and test teams to externally activate the UBO's as part of the testing procedure without employing any HCI just by using a simple procedure / function launch program (often called a test-bed or driver), along with a specially configured transient input-file (input test data) and prepared test database (initial state dataset such as that within a virtual environment or established through executing SQL scripts). These can be used to repeat the same set of testing, time and time again, as part of regression testing and re-test procedures. The expected outcome (i.e. results) would have been pre-prepared and validated to provide an 'oracle' against which the actual outcome is manually compared. The advantage of this approach is that the impact of the keyboard and mouse input upon the testing procedure is removed so that the UBO's can be tested using a standard set of repeatable test procedures; furthermore any changes made by the programmers to the HCI's would have no effect upon the associated UBO's.

This is especially useful for load and performance testing as multiple UBO's can be started simultaneously over a network of several computers to test multi-user operation and database volumes. This can be achieved, again by using minimal code, to construct a test-bed platform that will evoke the selected UBO's with appropriate transient input files and initial states at pre-determined times on different networked computers in either asynchronous or synchronous fashion as deemed required for the test. In addition, this approach can also be used to establish test environments for validating and verifying batch operations and overnight processing that can be simulated not only for a single day, but in fact for the entire projected life-time of a system. That is to say several years' worth of processing could be simulated with a set of controlled data to demonstrate that the system will operate as intended over that period. Indeed, spreadsheet simulations are excellent for calculating the expected outcome of such batch processing operations. Likewise database volumes could also be generated using multi-instance copy functions to replicate ancillary file records (i.e. lookup lists) and primary file records (i.e. customer, jobs and invoices, etc) by selecting the source record and specifying the number of replications from which the necessary database volumes can be generated. The resultant records would be same as the originals, however, they would be assigned their own unique ID's. Through the use of Database Volume the performance and capacity of the computers, networks and storage devices can be analysed accordingly. Thus regression testing / retesting, multi-user operation testing and load / performance testing can be conducted relatively cheaply without the need for expensive and complex tools such as Win-Runner and Load-Runner from Mercury Interactive (http://www-svca.mercuryinteractive.com/products/testing/). Another example of such tools would be Rational Suite Test Studio that includes Rational Robot which performs the same task as Win-Runner (see www.rational.com). The HCI components themselves can be tested using standard user interface testing techniques as defined by BS7925 part 2 to include equivalence partitioning and boundary value analysis. The only main draw-back with this approach is the added discipline and programming rigour required during its implementation, however, this is off-set by the 'added-value' provided to the testing and QA process. In fact this style of 'inbuilt-testing' can be expanded further to verify and validate the internal function and procedure calls contained within the various program code libraries. That is to say that the component testing of these internal function and procedure calls can also be automated using a similar technique which is very reminiscent of JUNIT, which is now being increasingly used in JAVA based eXtreme Programming. The basic premise is to establish a specialised test program that that will call each function and procedure in sequence using a set of hard coded test-data that has been programmed into the testing subroutine. The results of which can then be compared to the hard-coded expected results that the programmer or tester has prepared manually and verified as to the correct operation (i.e. an 'oracle' approach). This verification process can be embedded into the program as an automated function. It would be necessary to establish a separate 'test' database so that the function and procedure calls are independent of the live data (i.e. this prevents data corruptions and deletions) and prior to running each function and procedure call the initial-state of the test database may have to be re-established accordingly so that the results of current function or procedure is not affected by the past calls. If a function or procedure call parameter has multiple options then ideally each execution path should be tested to ensure a suitable full coverage (although this is not always practical or even

possible). The draw-back with this approach is that the test scripts have to be programmed into the software under test and that the automated testing may take a while to complete its task due to the test database being continually reverted back to the appropriate 'initial state' for each function and procedure call, however, this approach does provide reasonable assurances that the internal function and procedure calls are still functionally compliant and it can easily be run overnight by selecting either a menu option or by running an independent program that calls the testing subroutine.

The concept of 'inbuilt-testing' tools can be further expanded to cover Network Loading, Database Volume and Operational Performance as already discussed. That is to say for Network Loading a simple program can be written that would simulate 'user activity' so that the network traffic capacity and record-locking can be verified. Whereby for each simulated user a test program would be instigated on one of the networked computers that would randomly choose a database table and then again randomly choose an operation such as (1) to read one, several or all of the records; (2) randomly choose a record and lock it for several seconds; (3) randomly choose a record, lock it for several seconds and then write the data back; (4) do nothing for several seconds or even several minutes of inactivity. In fact, depending on the operating system, it would be possible to simulate a hundred or so ghost users from a couple of networked computers. The Network Loading and its subsequent knock-on-effect within the Operational Performance are then assessed accordingly in terms of timings, record-locking and network traffic for a range of potential users. Likewise, for Database Volume, 'auto-generation' of valid test data can be programmed that is randomly chosen according to pre-set criteria and patterns for a projected lifespan; the data from which can then also be used for training and sales purposes. The Database Volume and its subsequent knock-on-effect within the Operational Performance can then be assessed accordingly by varying the size of the generated database at predetermined intervals. It would be necessary to estimate the expected size of the database at the given predetermined intervals, or at the very least evaluate the system within the maximum perceived size of database. Again, for Operational Performance, this is essentially timings and known system boundaries. That is to say it is assessed if the speed and functional operation of the system is acceptable to the user given the various constraints of the system including the Network Traffic and Database Volume.

As part of this examination of semi-automated and inbuilt testing techniques, it is also recommended that the use of onscreen Check-Sums and onscreen Drill-Downs be promoted; particularly as it can be used to provide proof of compliance with Financial Governance and Corporate Due Diligence.

The concept being conveyed is to embed various 'check and balance' mechanisms within the system code so that the user can visually verify the system operation and for the system to automatically validate its own data. The additional development costs and timescales involved can easily be justified as the end result will provide added business value and product-confidence. Furthermore it is a particularly seductive sales feature for prospective clients and helps to satisfy the requirements of Governing Regulatory Bodies (such as the FSA) and the UK Legal system. To demonstrate the point, consider the concept of a Sales Ledger Reconciliation Report as shown in the example below …

*** Sales Ledger Reconciliation Report ***			
Sales Period	**M-T-D £**	**Y-T-D £**	**Cumulative £**
Sales Invoice	**£7,813.00**	**176,386.00**	**£373,179.58**
Sales Receipt	**-£2,000.00**	**-£134,793.00**	**£223,261.80**
Sales Credit Notes	**£0.00**	**£0.00**	**-£1,213.11**
Sales Debit Adjustment (+)	**£100.00**	**£12,125.00**	**£41,699.23**
Sales Credit Adjustment (-)	**£0.00**	**-£10,478.00**	**-£34,252.23**
Sales Discount	**-£701.00**	**-£14,865.50**	**-£22,347.48**
Totals	**£5,212.00**	**£28,374.50**	**£133,804.19**
Date Extracted	**30th June 2009**		

In the above case example, the Sales Ledger system provides a Reconciliation Report where the various transactions are analysed within a two dimensional matrix using a 'transactions type' verses 'defined periods' structure. Valid transaction types might be Sales Invoice, Sales Receipt, Sales Credit, Sales Adjustment (+/-) and Sales Discount. Likewise, valid 'defined periods' might be Month-to-Date, Year-to-Date and Cumulative. The purpose of the Reconciliation Report is simply to provide a snap-shot of the Sales Ledger at that specific moment in time.

From the perspective of the end-user, the figures displayed may or may not be correct, in truth they have no way of actually knowing and this can cause serious alarm and uncertainty; particularly if the history of the product has not been exemplarily. A design level approach to address these issues would be to display a series of checksums to verify the Reconciliation figures by calculating the Grand Total (Calculated Sales Leger Value) by various methods that should in theory generate the same value, however, from differing sources of information and using different computational algorithms.

*** Sales Ledger Reconciliation Checksums ***	
Calculated Sales Ledger Value from Reconciliation	£133,804.19
Sum of all Customer Account Balances	£133,804.19
Sum of all Outstanding Transaction Amounts (unallocated)	£133,804.19
Sum of all Original Transaction Amounts (excluding EOM Journals)	£133,804.19
Sum of Out-of-Phase Transaction Statement Balances	£133,804.19

Although such an approach does go a long way to address user concerns, it would be the ideal to incorporate an additional Drill-Down facility that can provide the user with a list of the corresponding Sales Ledger transactions that made up the Reconciliation Analysis figures. The Sales Ledger transaction listing would be generated using ad-hoc SQL scripts that are specific to the chosen cell combination. This facility provides added functionality that is of practical benefit to the end-user and would be instrumental in gaining their approval and trust of the system.

In the above Sales Ledger Reconciliation example, the value of the Sales Invoices for the Month-to-Date was **£7,813.00**. Using a Drill-Down facility, the list of transactions that made up that value might look like the following …

*** Invoice Drill Down List ***				
Date	Client	Transaction	Sale	Amount £
2nd June 2005	Mrs White	Invoice	Goods and Services	£1302.16
9th June 2005	Professor Plum	Invoice	Goods and Services	£1302.16
9th June 2005	Miss Scarlet	Invoice	Goods and Services	£1302.16
16th June 2005	Rev Green	Invoice	Goods and Services	£1302.16
23rd June 2005	Mrs Peacock	Invoice	Goods and Services	£1302.16
30th June 2005	Col. Mustard	Invoice	Goods and Services	£1302.20
Month / Year:	June 2009		Total:	£7,813.00

From the perspective of the user, to validate a particular Reconciliation figure, the corresponding cell would be double-clicked so that a Drill-Down is displayed (i.e. as shown above). This allows the end-user to view the full list of Sales Ledger transactions for that 'transaction type' and 'defined period' combination (i.e. Sales Invoice M-T-D). In particular a check-sum total would be provided at the end of the list that should match the corresponding Reconciliation total displayed within the matrix (e.g. **£7,813.00** in the above example). A further Drill-Down could then be provided to display the contents of Invoices and Credit Notes that are listed. For example, the transaction dated 16th June 2005 could have been selected and this would show the Invoice for Rev Green to the value of **£1302.16**.

To manually verify a handful of the Reconciliation figures would be a quick and simple task for the user, however, to verify all of the Reconciliation figures would be a time-consuming and laborious task. To verify the figures on a set of Productivity and Profitability Reports, which may have many thousands of analysis fields and records, would be most impractical indeed and may even be impossible to achieve by human means. It would, however, be viable to provide an automated tool within the software itself to undertake this task, particularly if it employs a functionally compliant but dissimilar mechanism that has also been verified and validated as to its operation. For example the main reporting mechanism may use generated Analysis tables stored within the database and the alternative reporting mechanism may use the SQL scripts as used in the Drill-Down listings. For the record, Analysis tables are extremely efficient in terms of performance, however, they are susceptible to issues caused by changes to source records (hence needing the occasional analysis file rebuild). Whereas, SQL scripts are always accurate, however, their performance is seriously deteriorated due to the volume of computations required to be performed.

Further still, the various Analysis and Extraction Reports produced by the system could be designed to be matched and cross matched together, particularly if the data being presented is derived from differing reporting perspectives and information sources. Hence if all is well then they should yield the same set of counts and totals. This match and cross-match of counts and totals provides added confidence in the operation of the system; better still if this verification process can be automated then this makes the software system more likely to be accepted by the end-users.

All of these approaches are not new; in fact their history can be traced back to the days of COBOL based main-frame (legacy) systems of the 60's and 70's, however, they are still applicable to modern day systems and their use is proactively encouraged by the author.

Briefing on the source data for Analysis and Statistics reports.

The source data for Analysis and Statistics reports can either be generated on the fly (e.g. using SQL scripts) or retained within specifically designated database tables. Both are appropriate but they do have their own particular set of advantages and limitations; hence they would be used in different circumstances and the main distinguishing factor being the volume and complexity of database records to be processed. In the case of reports generated on the fly the data is always current but the generation process is often extremely slow due to the volume and complexity of processing required. Whereas in the case of database tables then the data is often historical (hence the need for date / time stamps to be included within the reports to provide a chronological datum) but the performance is dramatically improved; this is particularly the case for large database volumes. Other issues with updating of designated database tables used for recording Analysis and Statistics is that they will require a record locking mechanism during update (usually prohibiting other users and system functions from also updating whilst processing) and that their content will be significantly impacted by any changes to ancillary files (particularly those used for groupings and columns on reports). Hence the designated database tables used for recording Analysis and Statistics will need regular periodical updates and the occasional rebuild; typically conducted as part of the overnight or end of month processing.

- As an example of using specifically designated database tables, Production and Profitability Analysis reports would typically be compiled from Invoices that have been printed and posted to the Sales Leger. In conjunction with specifically designated database tables an additional Analysed flag within the Invoices would indicate that they have already been included (i.e. preventing duplicate invoicing analysis). In the case of an Update function, which is periodic and incremental in nature, the specifically designated database tables would be updated with the details of any suitable unprocessed Invoices and the Analysed flag within the analysed Invoices would be duly assigned to prevent their future inclusion. In the case of the Rebuild function, which is occasional and all-encompassing in nature, the Analysed flags within the Invoices would first be cleared and the content of the specifically designated database tables would be deleted prior to the execution of the Update function. Essentially the Rebuild function is the Update function, however, preliminary actions are conducted that first clear down the specifically designated database tables and then clear any Analysed flags within the Invoices. The content of Production and Profitability Analysis reports would typically consist of product item listings depicting Product ID, Description, Quantity, Cost, Profit and Sell which may optionally be grouped by report headings (such as Customer, Staff, Department, Job, Invoice, etc) or sub-groupings (such as Customer Group, Staff Group, Department Group, Job Type, Invoice Type, etc). Other fields could be included such as depiction of effort-required / work-undertaken (such as Units or Hours) and possibly Rework Analysis depicting any failed work that had to be redone at discount (or free of charge) and the associated cost to the business as a result.

- As an example of using on the fly generation (e.g. using SQL scripts), Financial Reconciliation Analysis reports within a Sales Leger would typically be compiled from posted account transactions. These reports may take some time to compile even though the resultant transaction dataset is often a smaller than that generated for the Production and Profitability Reports, however, the final results would be current and would not affect other users during compilation. The content of Financial Reconciliation Analysis reports would typically consist of a matrix depicting Transaction Types against standard Periodicals with Totals. Examples of Transaction Types being Sales Invoices, Sales Receipts, Sales Credits, Sales Debit Adjustments (+), Sales Credit Adjustments (-) and Sales Discounts. Examples of standard Periodical being Month to Date (M-T-D), Year to Date (Y-T-D) and Cumulative (i.e. all years and months to-date).

Briefing on Test Oracles and their application

'Oracles' are usually either specialised software tools or previous versions of the software that are known to operate without issue and whose operation does not deviate from the system specification. These 'Oracles' are then used to generate input data and / or expected results that are intended to validate and verify the software under test. 'Oracles' may take several forms, these include …

- Parallel functionality such as competitor's products, previous versions, openly documented industry models and formal standards. Also the system functionality can be designed to be 'overlapped' so that critical operations such as the analysis reports and batch processing can be verified and validated by employing more than one dissimilar mechanism to produce the same result. Although some might consider this wasteful, it does provide a reliable and effective approach to testing 'live' systems that are currently in active use. Particularly as live systems must not be entered with test data as it would affect the integrity of the data held on file. This is particularly the case for Accounting and Financial systems.

- Internal Safety-Checks and Check-Sums. Where by the software is programmed to conduct frequent internal checks and checksums to ensure system integrity. Typical checks would be to verify the validity of system flags, totals, counts, balances and foreign key relationship links, etc.

- Mathematical Rules (e.g. sum of angles in a triangle is 180 degrees). In some cases the results of the system can be pre-determined by performing a series of mathematical rules and formulae. This approach can be extended to the development of automated data generation facilities where by 'patterned' test data can be randomly generated according to set criteria, so that the appropriate volumes of data can be created for performance and load which can be readily checked in terms of its content.

- Saved results from previous tests (i.e. used to validate and verify operation, consistency and regression compliance). This is conducted as part of standard pre-release testing procedures, where by past tests are regularly repeated within a retesting and regression testing policy. Ideally the test data and expected results should be hard coded and benchmarked against a known 'oracle' or 'trusted source'.

- Pattern matching is an extremely effective tool for testing the underlying mathematics within large databases, reports and views, etc. In which a set of easily recognisable test data (usually all 1's, 2's, 5's and 10's or other simplistic figures) is employed along with their multiples so that the results of calculations can be readily checked visually without the need of a calculator or computational spreadsheet. Likewise expected results can be distinctively patterned accordingly so that unexpected or adverse results can be readily identified upon visual inspection. Essentially so long as the repetition is consistent and corresponding to the expected pattern then all is well.

Briefing on Rounding Errors

The definition of “Rounding Errors” is defined as ...

A round-off error, also called rounding error, is the difference between the calculated approximation of a number and its exact mathematical value. Numerical analysis specifically tries to estimate this error when using approximation equations or algorithms, especially when using finite digits to represent infinite digits of real numbers. This is a form of quantisation error.

Source: http://en.wikipedia.org/wiki/Round-off_error

In the case of computer software the calculated approximation being the individual line item values that are rounded to a given decimal place (typically 2DP) and the total summated value (which is also usually rounded to the same given DP). However, each individual item may be stored to several DP depending upon the memory data type being used to store the financial information (e.g. the ‘Decimal’ data type used in VB can have up to 28 digits to the right of the decimal point). Where only a few items of small monetary value exist then there is usually little or no rounding error, however, where items are extensive with outsized monetary values and quantities involved then the rounding error can be rather excessive. For example, within large scale Construction projects (typically multi-million £ pound sterling), the rounding error can result in tens of thousands of pounds (or more) which needs to be accounted for. That is to say from the perspective of a third person it appears that money has literally disappeared into thin air! In reality it has it has simply not been disclosed.

Another likely candidate being ratio mathematics where fractural quotients are involved; particularly after multiplication and division of monetary values. For example UK pound sterling (£) is normally represented as 2 DP such as £1.23, however, the underlying calculated value might actually be 1.23456789!

Likewise rounding errors are often caused through the integration of several software systems where the data is passed between functionally compliant but independent system components using partial parameters; for example an Invoicing module may pass transaction data to a separate Accounts module using only the cost value (e.g. £300) with an uplift rate (e.g. 66.67%). If all being well then this should result in matching calculated values (i.e. £500.01) in both systems. However, if there is a rounding error then the results might be slightly different.

Thus to adjust the rounding error accordingly, it is necessary to present a mathematically balancing correction which is constituted from the difference between the calculated approximation and its exact mathematical value. This mathematically balancing correction can then be added as a final item that brings these two values together back into alignment. Alternatively this mathematically balancing correction can be split over several items using the mechanism of proportional redistribution using ratios to mitigate the impact of such a large correction value. On the following page is a sample spreadsheet, which although is only a very simple example with a rounding error of 7p, does clearly demonstrate the principals involved.

NB:

- DP represents the term 'Decimal Places'
- The ‘Decimal’ data-type represents the Ms Visual Basic 'Decimal' data type which is constituted by a 16 byte value in the range of 0 through +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) with no decimal point or 0 through +/-7.9228162514264337593543950335 with 28 digits to the right of the decimal. The smallest nonzero number is +/-0.0000000000000000000000000001 (+/-1E-28)

	Column-B	Column-C	Column-D	Column-E	Column-F	Column-G
Row 2						
Row 3			*** ***Rounding Errors Simulation*** ***			
Row 4						
Row 5	Quantity		1.23456789	<= User Input		
Row 6	Price		9.87654321	<= User Input		
Row 7	Decimal Places	(DP)	2	<= User Input		
Row 8						
Row 9	Quantity	Price	Precision	Formulae	Rounded	Formulae
Row 10	1.23456789	9.87654321	12.19326311	=C10*D10	12.19	=ROUND(D10,$D$7)
Row 11	1.23456789	9.87654321	12.19326311	=C11*D11	12.19	=ROUND(D11,$D$7)
Row 12	1.23456789	9.87654321	12.19326311	=C12*D12	12.19	=ROUND(D12,$D$7)
Row 13	1.23456789	9.87654321	12.19326311	=C13*D13	12.19	=ROUND(D13,$D$7)
Row 14	1.23456789	9.87654321	12.19326311	=C14*D14	12.19	=ROUND(D14,$D$7)
Row 15	1.23456789	9.87654321	12.19326311	=C15*D15	12.19	=ROUND(D15,$D$7)
Row 16	1.23456789	9.87654321	12.19326311	=C16*D16	12.19	=ROUND(D16,$D$7)
Row 17	1.23456789	9.87654321	12.19326311	=C17*D17	12.19	=ROUND(D17,$D$7)
Row 18	1.23456789	9.87654321	12.19326311	=C18*D18	12.19	=ROUND(D18,$D$7)
Row 19	1.23456789	9.87654321	12.19326311	=C19*D19	12.19	=ROUND(D19,$D$7)
Row 20	1.23456789	9.87654321	12.19326311	=C20*D20	12.19	=ROUND(D20,$D$7)
Row 21	1.23456789	9.87654321	12.19326311	=C21*D21	12.19	=ROUND(D21,$D$7)
Row 22	1.23456789	9.87654321	12.19326311	=C22*D22	12.19	=ROUND(D22,$D$7)
Row 23	1.23456789	9.87654321	12.19326311	=C23*D23	12.19	=ROUND(D23,$D$7)
Row 24	1.23456789	9.87654321	12.19326311	=C24*D24	12.19	=ROUND(D24,$D$7)
Row 25	1.23456789	9.87654321	12.19326311	=C25*D25	12.19	=ROUND(D25,$D$7)
Row 26	1.23456789	9.87654321	12.19326311	=C26*D26	12.19	=ROUND(D26,$D$7)
Row 27	1.23456789	9.87654321	12.19326311	=C27*D27	12.19	=ROUND(D27,$D$7)
Row 28	1.23456789	9.87654321	12.19326311	=C28*D28	12.19	=ROUND(D28,$D$7)
Row 29	1.23456789	9.87654321	12.19326311	=C29*D29	12.19	=ROUND(D29,$D$7)
Row 30						
Row 31	Totals ∑	Precision	243.8652622		N/A	
Row 32	Totals ∑	Rounded	243.87		243.8	
Row 33						
Row 34	Differences:	Between $F$30 and $D$29			0.0652622	
Row 35						
Row 36		Between $F$30 and $D$29 to 2DP			0.07	
Row 37						

There are other mechanisms for rounding errors which incur greater inaccuracies; for example the mechanism whereby the Quantity and Price are first Rounded to 2DP (i.e. Accountancy format) before being multiplied together. This approach allows for the apparent rounding error to be masked but not resolved as depicted within the following example spreadsheet that results in a value of £243.05 and not £243.80; in other words there is a 75p difference which is a very significant inconsistency.

	Column B	Column C	Column D
Row 2			
Row 3	**Cost Items …**		
Row 4			
Row 5	Quantity:	1.23456789	
Row 6	Price:	9.87654321	
Row 7	Line Iterations:	20	
Row 8	Quantity (2DP):	1.23	=ROUND(C5,2)
Row 9	Price (2DP):	9.88	=ROUND(C6,2)
Row 10	Line Total (Unrounded):	12.1524	=C8*C9
Row 11	All Lines Total (2DP):	243.05	=ROUND(C10*C7,2)

That is to say, the inputs applied within the Job Costs being …

- Item Line 1 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 2 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 3 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 4 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 5 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 6 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 7 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 8 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 9 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 10 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 11 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 12 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 13 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 14 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 15 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 16 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 17 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 18 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 19 having a Quantity of 1.23456789 with a Price of £9.87654321
- Item Line 20 having a Quantity of 1.23456789 with a Price of £9.87654321

Of the two extremities presented it is better to employ the mechanism within the first example as this keeps the totals correct. Any ‘Rounding Adjustment’ should be included within the VAT Analysis on the Invoice Net £ but would ideally be presented for inspection within an additional column. Also to help mitigate rounding errors it is necessary to ensure that the input fields for QUANTITY and PRICE are restricted to suitable decimal places in accordance with the monetary values being used (e.g. 2DP for GB Pounds Sterling and also 2DP would cater for most Quantities). Furthermore the number of detail lines should kept to the very minimum where possible; ideally no more than twenty items.

On the next page is a sample VAT Analysis table intended for use on Quotes, Job Orders and Invoices (both Sales & Purchase) …

VAT ANALYSIS						
Tax Code	**Description**	**Rate %**	**Net £**	**VAT £**	**Gross £**	**Rounding Adjustment**
A	**Standard**	**17.5 %**	**£ 1000**	**£ 175**	**£ 1175**	**0.23**
B	**Fuel**	**5 %**	**£ 2000**	**£ 100**	**£ 2100**	**0.04**
C	**Zero**	**0 %**	**£ 550**	**£ 0**	**£ 550**	**0.00**
D	**Except**	**0 %**	**£ 450**	**£ 0**	**£ 450**	**0.00**
Totals			**£4000**	**£275**	**£ 4275**	

NB: Within the Quote, Job Orders and Invoice lines only the Net £ amount and VAT code needs to be recorded against each line as the VAT Analysis includes the all necessary totals and rate disclosure.

However, there are occasions when the Rounding Error is so large that it cannot be included within the VAT Analysis as a separate Adjustment column; it would simply not be acceptable to the client receiving the invoice! In such circumstances it is necessary to proportionally redistribute the resultant 'Rounding Adjustment' within the Invoice content. Every penny is still accounted for but the Rounding Adjustment applied is not publicly disclosed.

The spreadsheet on the following page demonstrates such mathematics involved; indeed the same technique is also used to adjust estimate values within bid tenders so that the finale figure tallies with the target imposed. In the example given, ten items to the value of £550 are proportionally redistributed using ratios so that there adjusted total value now represents £660 which is an increase of £110.

Proportional Ratio Adjustment Formulae:

Revised Item = Revised Total * (Original Item / Original Total)

Proportional Ratio Adjustment Derivation:

Revised Item / Revised Total = Original Item / Original Total

NB: * represents multiplication and / represents division

	Column B	Column C	Column D	
Row 2				
Row 3	Inputs	£		Formulae
Row 4				
Row 5	Original Item #1		10	<= USER INPUT
Row 6	Original Item #2		20	<= USER INPUT
Row 7	Original Item #3		30	<= USER INPUT
Row 8	Original Item #4		40	<= USER INPUT
Row 9	Original Item #5		50	<= USER INPUT
Row 10	Original Item #6		60	<= USER INPUT
Row 11	Original Item #7		70	<= USER INPUT
Row 12	Original Item #8		80	<= USER INPUT
Row 13	Original Item #9		90	<= USER INPUT
Row 14	Original Item #10		100	<= USER INPUT
Row 15				
Row 16	Original Total		550	=SUM($C$5:$C$14)
Row 17				
Row 18	New Adjusted Total		660	<= USER INPUT
Row 19				
Row 20	Revised Item #1		12	= $C$18 * ($C$5/$C$16)
Row 21	Revised Item #2		24	= $C$18 * ($C$6/$C$16)
Row 22	Revised Item #3		36	= $C$18 * ($C$7/$C$16)
Row 23	Revised Item #4		48	= $C$18 * ($C$8/$C$16)
Row 24	Revised Item #5		60	= $C$18 * ($C$9/$C$16)
Row 25	Revised Item #6		72	= $C$18 * ($C$10/$C$16)
Row 26	Revised Item #7		84	= $C$18 * ($C$11/$C$16)
Row 27	Revised Item #8		96	= $C$18 * ($C$12/$C$16)
Row 28	Revised Item #9		108	= $C$18 * ($C$13/$C$16)
Row 29	Revised Item #10		120	= $C$18 * ($C$14/$C$16)
Row 30				
Row 31	Check Total		660	=SUM($C$20:$C$29)
Row 32				

Briefing on Quotes, Job Orders and Invoice Mathematics (Sales & Purchases)

The formulae for Quotes / Job Orders / Invoice Mathematics and VAT Mathematics are essentially the same; with Discount Mathematics having a slight variation in formulae. They have been included as they provide a useful QA benchmark.

Job / Invoice Mathematics …

Calculate Cost £ from Sell £ and UpLift % …

	Column-B	Column-C	Column-D
Row 1			
Row 2	Inputs …		
Row 3			
Row 4	Item.1 - Selling Price Amount £	1175	
Row 5	Item.2 - Uplift %	17.5	NB: enter positive sign value
Row 6			
Row 7	Results …		
Row 8			
Row 9	Item.3 - Profit £	175	= $C$5 * ($C$4/(100+$C$5))
Row 10	Item.4 - Cost Price Amount £	1000	= ($C$4 - $C$9)
Row 11			
Row 12	Alternative usage: Calculate Cost Price Amount £ from Selling Price Amount £ and Uplift % (positive sign value)		

Calculate Sell £ from Cost £ and UpLift % …

	Column-B	Column-C	Column-D
Row 1			
Row 2	Inputs …		
Row 3			
Row 4	Item.1 - Cost Price Amount £	1000	
Row 5	Item.2 - Uplift %	17.5	NB: enter positive sign value
Row 6			
Row 7	Results …		
Row 8			
Row 9	Item.3 - Profit £	175	=($C$5/100)*$C$4
Row 10			
Row 11	Item.4 - Selling Price Amount £	1175	=$C$4 + $C$9
Row 12			
Row 13	Alternative usage: Calculate Selling Price Amount £ from Cost Price Amount £ and Uplift % (positive sign value)		

Calculate UpLift% from Cost £ and Sell £ …

	Column-B	Column-C	Column-D
Row 1			
Row 2	Inputs …		
Row 3			
Row 4	Item.1 - Cost Price Amount £	1000	
Row 5	Item.2 - Selling Price Amount £	1175	
Row 6			
Row 7	Results …		
Row 8			
Row 9	Item.3 - Uplift %	17.5	=(($C$5-$C$4)/$C$4)*100
Row 10			
Row 11	Alternative usage: Calculate Uplift % from Cost Price Amount £ and Selling Price Amount £		

Briefing on Quotes, Job Order and Invoice Mathematics (Sales & Purchases) – Cont'd

The formulae for Quotes / Job Orders / Invoice Mathematics and VAT Mathematics are essentially the same; with Discount Mathematics having a slight variation in formulae. They have been included as they provide a useful QA benchmark.

VAT Mathematics …

Calculate VAT exclusive Invoice Amount £ from VAT inclusive Invoice Amount £ and VAT Rate % …

	Column-B	Column-C	Column-D
Row 1			
Row 2	Inputs …		
Row 3			
Row 4	Item.1 - VAT inclusive Invoice Amount £	1175	i.e. Gross £
Row 5	Item.2 - VAT Rate %	17.5	NB: enter positive sign value
Row 6			
Row 7	Results …		
Row 8			
Row 9	Item.3 - VAT Amount £	175	= $C$5 * ($C$4/(100+$C$5))
Row 10	Item.4 - VAT exclusive Invoice Amount £	1000	= ($C$4 - $C$9)
Row 11			
Row 12	Alternative usage: Calculate Cost £ from Sell £ and UpLift % (positive sign value)		

Calculate VAT inclusive Invoice Amount £ from VAT exclusive Invoice Amount £ and VAT Rate % …

	Column-B	Column-C	Column-D
Row 1			
Row 2	Inputs …		
Row 3			
Row 4	Item.1 - VAT exclusive Invoice Amount £	1000	i.e. Net £
Row 5	Item.2 - VAT Rate %	17.5	NB: enter positive sign value
Row 6			
Row 7	Results …		
Row 8			
Row 9	Item.3 - VAT Amount £	175	=($C$5/100)*$C$4
Row 10			
Row 11	Item.4 - VAT inclusive Invoice Amount £	1175	=$C$4 + $C$9
Row 12			
Row 13	Alternative usage: Calculate Sell £ from Cost £ and UpLift % (positive sign value)		

Calculate VAT Rate % from VAT exclusive Invoice Amount £ and VAT inclusive Invoice Amount £ …

	Column-B	Column-C	Column-D
Row 1			
Row 2	Inputs …		
Row 3			
Row 4	Item.1 - VAT exclusive Invoice Amount £	1000	i.e. Net £
Row 5	Item.2 - VAT Inclusive Invoice Amount £	1175	i.e. Gross £
Row 6			
Row 7	Results …		
Row 8			
Row 9	Item.3 - VAT Rate %	17.5	=(($C$5-$C$4)/$C$4)*100
Row 10			
Row 11	Alternative usage: Calculate UpLift% from Cost £ and Sell £ (positive sign value)		

Briefing on Quotes, Job Order and Invoice Mathematics (Sales & Purchases) – Cont'd

The formulae for Quotes / Job Orders / Invoice Mathematics are essentially the same; with Discount Mathematics having a slight variation in formulae. They have been included as they provide a useful QA benchmark.

Discount Mathematics …

Calculated Discounted Pricing £ from List Pricing £ and Discount Rate % …

Row 1	Column-B	Column-C	Column-D
Row 2	Inputs …		
Row 3			
Row 4	Item.1 - List Price £	1175	
Row 5	Item.2 - Discount Rate %	17.5	NB: enter positive sign value
Row 6			
Row 7	Results …		
Row 8			
Row 9	Item.3 - Discount Amount £	175	= $C$5 * ($C$4/(100+$C$5))
Row 10	Item.4 - Discounted Price £	1000	= ($C$4 - $C$9)
Row 11			
Row 12			

Calculate List Pricing £ from Discounted Pricing £ and Discount Rate …

Row 1	Column-B	Column-C	Column-D
Row 2	Inputs …		
Row 3			
Row 4	Item.1 - Discounted Price £	1000	
Row 5	Item.2 - Discount Rate %	17.5	NB: enter positive sign value
Row 6			
Row 7	Results …		
Row 8			
Row 9	Item.3 - Discount Amount £	175	=($C$5/100)*$C$4
Row 10			
Row 11	Item.4 - List Price £	1175	=$C$4 + $C$9
Row 12			
Row 13			

Calculate Discount Rate % from List Pricing £ and Discounted Pricing £ …

Row 1	Column-B	Column-C	Column-D
Row 2	Inputs …		
Row 3			
Row 4	Item.1 - Discounted Price	1000	
Row 5	Item.2 - List Price £	1175	
Row 6			
Row 7	Results …		
Row 8			
Row 9	Item.3 - Discount Rate %	17.5	=(($C$5-$C$4)/$C$4)*100
Row 11			

Briefing on 'off-the-shelf' SME Accounts Software

Some systems include their own Accounting functions, which are often limited in scope and generally focus upon a Sales Ledger that is specific to the industry sector; possibly including a Purchase Ledger for additional procurement options. The problem being that their legality, functionality and stability may be called in to question if their operational remit is too narrow or too vague or their historical track record has not been extemporary. In such circumstances it is best to focus on the export of Invoices to 'off-the-shelf' Accounting software packages and to ensure that their financials tally exactly at both the point of export and import; this means the associate responsibility can also be offloaded (aka externalised). Indeed it can be rightly argued that the Accounting functions should be kept separate from the front-line operations and that access should be restricted to authorised persons only using specifically designated PC's that are appropriately secured from external threats.

Within the UK, the main options for 'off-the-shelf' SME Accounting software packages being predominately SAGE and QUICKBOOKS, there are other highly commendable software packages but these are the two products being considered here.

Every Book-Keeper and Accountant has their own particular preference; some are SAGE aficionados and others are devotees of QUICKBOOKS. Indeed both choices are justified as these SME software packages are established and professionally endorsed in the UK and worldwide; they are applicable to most industry sectors and include the necessary data import facilities or middleware so that financial information and Account transactions can be sourced from specialist third party software.

Regardless of the choice of Accounts software package, the system employed should conduct proper book keeping to legally recognised financial standards and industry best practices; this operation should be periodically validated and externally verified by appropriately qualified individuals including QA and Accounts auditors, as well as, senior company officials. Proprietors and directors of companies are legally and financially held responsible for their businesses including the software they use and the consequences of any issues arising as a result.

Some software vendors claim that 'off-the-shelf' SME Accounting software packages such as SAGE and QUICKBOOKS do not reflect the specialisation of their particular industry sector and are either too generic or too complex for practical usage. This may be the case for some, but for most this is highly unlikely and indeed is often just a blatant sales ploy! Only large scale organisations using ERP and CRM systems would need their own dedicated financial accounting packages.

From the authors experience of using 'off-the-shelf' SME Accounting software packages, both SAGE and QUICKBOOKS can be summarised as being tried, tested, documented, stable, supported and operates as per specification (as defined by product manuals and advertising). However, there will always be a need for specialist third party software that provides the features and functions not offered by standard 'off-the-shelf' SME Accounting software packages; for example Manufacturing, Construction, Engineering and Project Management Consultancy are all typical candidates.

Where specialised software is required by the SME then it should provide the necessary Account transaction and Invoice export to external 'off-the-shelf' Accounting software packages. Indeed additional middleware can be purchased that allows for import of transactions and invoices from standard CSV files (i.e. ASCII Comma Separated Values format). For example ADEPT provide a CSV Transaction Import for SAGE and Zed AXIS provide a CSV Transaction Import for QuickBooks; both of which can import multi-line VAT based invoice transactions and correctly associate their links between the invoice and their associated detail lines.

Terminology …

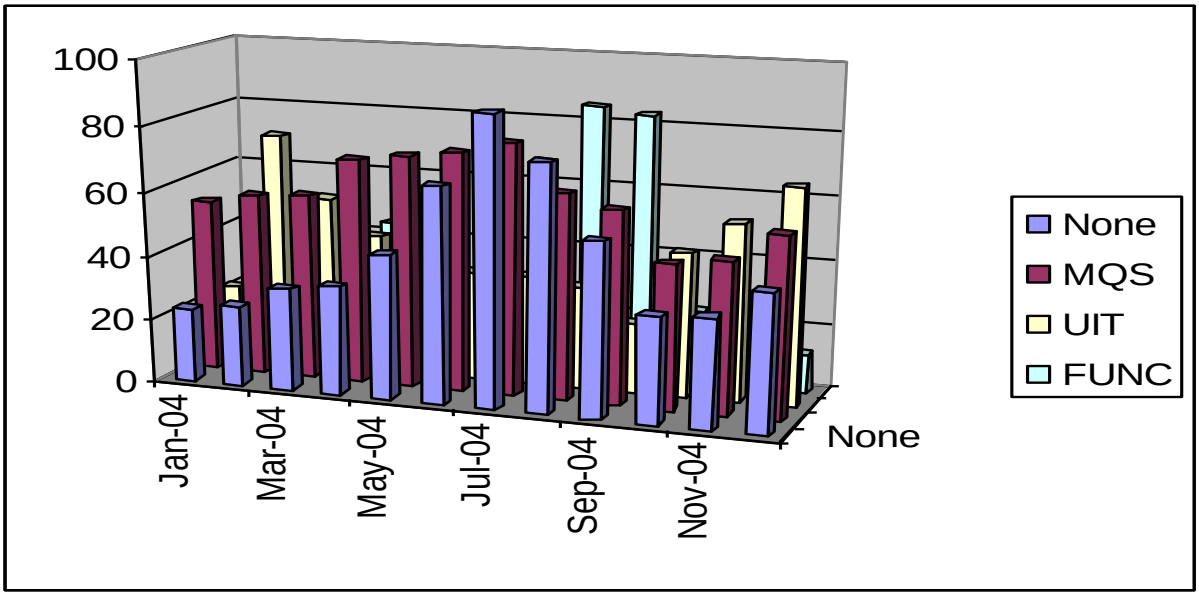
- SME: Small to Medium Enterprises
- ERP: Enterprise Resource Planning
- CRM: Customer Relations Management

Briefing on Software Quality Metrics

The use of Software Quality Metrics is far from new, in fact for many years it has been highly recommended as being good practice within the Computer Industry. Software Quality Metrics are most useful when they have been base-lined at regular intervals and the statistical trends analysed, however, it may take several weeks of data sampling before the results are accurate and reliable. Although, once this initial 'tooling-up' period is complete, all the effort and expense invested does soon repay itself many times over by highlighting the various 'weak areas' within the software development and testing, as well as providing official statistics that can be used to prove the quality of the software.

The Software Quality Metrics can be as simple or as complex as deemed necessary. For example analysing logged issues by status, priority and manpower resource is very effective for tasking as it focuses attention on what is required to be tackled. Alternatively section 4 of BS7925 part two contains extensive details of various Software Quality Metrics that can be employed within a micro environment (i.e. at the programming level), however, such metrics can be difficult to compile and are prone to people's differing interpretations and personal agendas.

Software Quality Metrics can be presented to clients and their end-users alike in order to gain both their trust and support in the delivery of new or upgraded products. In doing so good relations and product confidence can be steadily built upon solid foundations that can then be appropriately managed to drive forward sales and product development. Likewise valuable cost / time / manpower statistics can also be extracted that provides a reliable benchmark when compiling estimates for future testing activities.



There are no fixed rules in the use of Software Quality Metrics for the macro environment (i.e. Quality Management System) other than that they should be both applicable to the product and simple for people to understand. For example, various ratios could be used to depict the amount of new testing, retesting and regression testing undertaken. This could then be further categorized by the level of testing that was applied such as Minimum Quality Standard, User Interface Testing and Functional Testing. In addition extra complementary statistics could be included such as the number of Caveats (i.e. warnings) that were applied to the software releases along with the analysis of the growth rate of the software products in terms of their size / complexity, functionality and test coverage. All of which should be sampled at regular intervals to provide trends analysis based on daily, weekly, monthly, quarterly and yearly figures that can then be presented as either tabulated lists or pictorial graphs using actual or percentage values as deemed appropriate. Graphs based on percentage values are generally considered to be extremely useful as they can compare trends analysis based on more than one series et and can be readily interpreted by most people. Furthermore, compared to a plain tabulated listing, the use of a graph provides visual interest and a focus for attention.

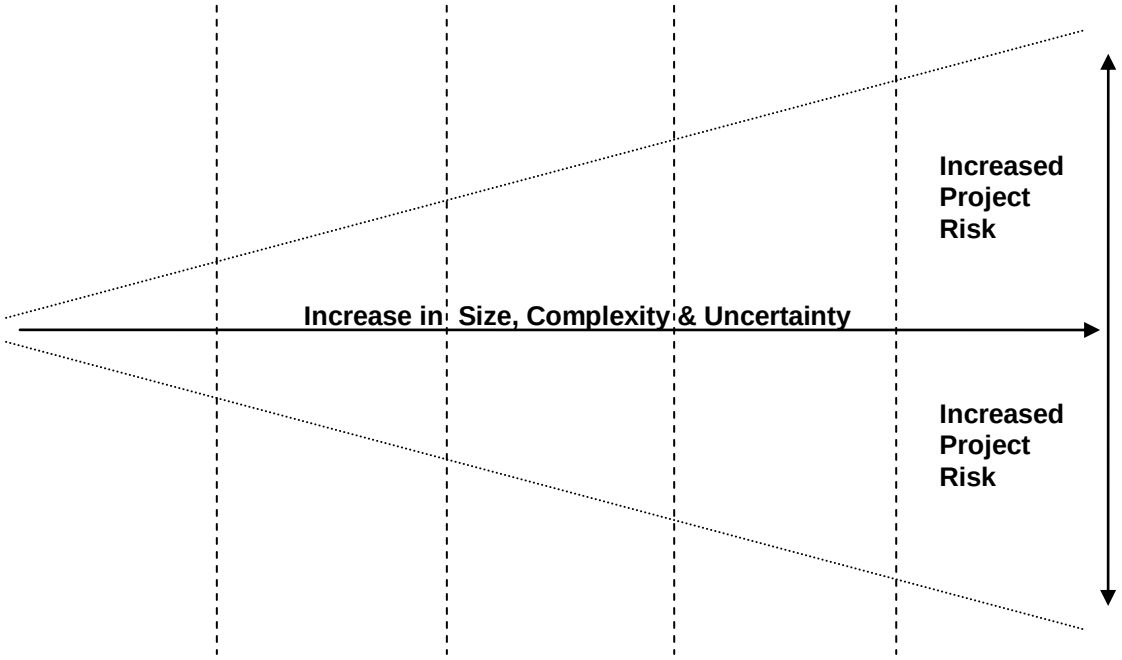
The list of possibilities for Software Quality Metrics are virtually endless, however, this is never always practical nor desirable as it can cause confusion and the production of Software Quality Metrics can also be costly in terms of time, effort and skills required. Hence it is important to rationalise Software Quality Metrics appropriately for effective use and one may wish to employ an appropriate software tool for the capture and tracking of support / testing activities from which statistical trends analysis can be extracted. For example, Test Track Pro and Visual Intercept are two such examples of call logging / support systems that are available that provide trends analysis:

- Test Track Pro from Seapine Software (TTPro) - http://www.seapine.com/ttpro.html
- Visual Intercept from Elsinore (VI) - http://www.elsitech.com

It is important to remember that whatever Software Quality Metrics are used, it must be 'quantifiable' and above all comprehendible so that the results can be directly presented and compared by external parties including the senior management, the customers and their end-users. In fact the client's clientele (or their legal representatives) may also wish for documented assurances to be made as to the operational status of the system for which Software Quality Metrics can provide the desired answers being sought.

Briefing on Risk Involved in Software QA and Software R&D

Risk increases as Project Size, Complexity and Uncertainty increases …



1. Green	2. Yellow	3. Orange	4. Red	5. Grey
OK	Doable	Caution	Danger	Money Pit!

- Sector 1 (green) has the least risk; whereas, sector 5 (grey) has the most risk.
- Offshore outsourcing and UK contracting of software testing and its associated quality assurance would be mostly located at sectors 1 (green) and 2 (yellow).
- Conversely, if a project is of significant scale or poorly managed, then the software development and its testing can often end up located at sectors 4 (red) and 5 (grey); regardless whether conducted in-house or outsourced.
- Sectors 4 (red) and 5 (grey) could be justified if the intention is to expand in to other vertical markets or increase the customer sales in the current market sector. The intention being that this would produce the necessary Return on Investment (ROI) required to recover the cost incurred. The caveat being that the associated technicalities and practicalities involved will have to be overcome as a matter of priority resulting in increased manpower requirements, deadlines and budgets with potentially no guarantee of success.
- In order to help minimise the risk with software testing and its associated quality assurance, the mandatory materials required would need to include the full set of test scripts accompanied by the technical documentation including system specifications and manuals.
- Likewise, to help minimise the risk with software development, the additional materials required would include existing product versions and working prototypes along with their associated source code. The functional operation of which can then be evaluated accordingly to provide a benchmark for a comparison with the new systems under development.

Briefing on System Configuration Permutations

It is a mandatory prerequisite to identify ALL of the components that make up the various system configurations; for example operating system, database, browser, third-party software and network configurations, etc.

This information is then used for cost estimation of the time and manpower requirements that takes into account the testing of each system function against each of the chosen system configurations through the use of an appropriate test script. This serves two purposes, firstly to exercise system functionality and secondly to gauge the performance timings which can then be used as a historical benchmark.

It is normal practice to test the software within several standard system configurations; using the example of a typical PC application then the test configurations might include the following options:

- Microsoft Windows (x4 options): XP, VISTA, Windows 7-32bit or Windows 7-64bit.
- Database (x2 options): Microsoft Access or Microsoft SQL Server.
- Internet Browsers (x6 options): Apple Safari, Google Chrome, Ms Windows Internet Explorer, Mozilla Firefox or Opera either separately or collectively (i.e. All Internet Browsers).
- Third-party software (x2 options): Ms Office (Word / Excel) or Preview Browsers.
- Network Configuration (x3 options): Single User (Stand Alone PC), Small PC Network (Multi-User using free MSDE / Express version database or Large PC Network (Multi-User using full version of Ms SQL Server database).

In the case of the above example it can be seen that there are 288 possible system configuration options. This figure was calculated from four windows operating system options multiplied by two database options multiplied by six browser options multiplied by two third-party software options multiplied by three network configuration options.

It would NOT be commercially viable to test all possible combinations and so a process of rationalisation is required so that the base set of likely system configurations are identified and these would officially nominated for testing purposes.

For example the extremities of the configuration range are depicted below; within which the remaining 286 combinations would reside…

- Top of the customer range would employ Windows 7-64, Microsoft SQL Server, All Internet Browsers, Ms Office and Multi-User.
- Bottom of the customer range would employ Windows XP, Microsoft Access, Windows Internet Explorer, Browser Previews and Single-User.

Both of these extremities would need to be tested along with several other interim combinations that provide best coverage of the options for operating system, database, internet browser, third-party software and network configuration. Obviously market research is required to determine the likely client installation. Alternatively it would be better to standardise on a specific configuration where the mandatory requirements would be Microsoft SQL Server, Ms Windows Internet Explorer, Ms Office and Multi-User; thus only leaving the option of operating system available to the client and so considerably restricting the number of combinations to be tested.

Briefing on the Roles and Responsibilities of Software Testers and QA Analysts

The roles and responsibilities of Software Testers and Quality Assurance (QA) Analysts are the essentially same, however, QA Analysts are also tasked with producing Test Scripts and associated Technical Documentation. As a guide the following has been compiled from typical recruitment advertisements and published job specifications for the QA Analyst role:

Remit …

- To undertake rigorous quality assurance testing of software releases prior to despatch.
- To formulate and maintain test plans, test scripts and test data to assist in the QA process.
- To update the Helpdesk system with progress of software testing and to generate release notes for new releases.
- To provide cover for the Helpdesk (telephone and email support) and Software Development teams when required.
- To research issues found with the software and to ascertain their nature (i.e. data problem, code bug, enhancement request or user misuse / misconception), and to identify the required fix (in conjunction with the Development and Support teams as necessary). Where mandated to do so, this may include tracing through the source code of the software product and undertake appropriate investigation in order to postulate the likely cause of the reported issues. Where mandated to do so, to undertake modification and enhancement of the source code for the software products under the direction of the Software Development team.
- Where appropriate to effect data fixes at customer sites (via remote access or on-site visit) and for more complex fixes, ensure that these are carried out by the assigned Software Developer.
- To prioritise and plan the work carried out by the assigned Software Developer(s) concerning the reported issues and to chase their progress.
- To update on-line help and user documentation with details of any changes implemented within the program.

Key Result Indicators ...

- Develop a detailed knowledge of software packages being supported in order to answer customers' enquiries and assist them in solving problems. Provide cover for the telephone support desk as required and ensure customers are kept regularly updated as to the status of their calls.
- If unable to resolve the issues then categorise and prioritise the issues and allocate them to the Software Development team and / or with Software Support team. Where necessary log and track the issues on appropriate computerised systems for issue resolution management. Any issues that are not being dealt with quickly enough should be escalated and the chasing of fixes should be proactive.
- Work closely with the Software Development and Support teams to ensure prompt fixing of software bugs.
- Maintain a log of testing activities from which Software Quality Metrics can be analysed and reported upon.
- Production and maintenance of product support material such as release notes and user manuals that are appropriate for the target audience and that are of a suitable professional quality.

Required Personal Attributes …

- Focussed, analytical and fastidious self-starter with the tenacity to quickly get to the core of a problem.
- Experienced of software quality assurance testing and formation of test scripts, test data and expected outcome.
- Committed to the company working procedures and compliance to industry standards (including ISO 9001, BS7925, IEEE 829 and ISEB syllabuses).
- Possesses a strong sense of personal responsibility, accountability and openness.
- Computer literate with working knowledge of Ms Windows (and where appropriate Linux) based software packages and operating systems. Ideally with an Information Technology or Computing related degree.
- An ability to quickly learn within a technical environment and to enforce the highest quality standards of work.
- Ability to work independently whilst possessing the interpersonal skills to liaise effectively with other members of the team and with customers.
- Excellent communication skills (both written and verbal – including telephone).
- Ideally formal training in QA testing standards (e.g. ISEB) and knowledge of GUI / Load / Performance automated testing tools would be advantageous.

Briefing on Recruitment and Diversity Policies

All CV's and communications from candidates must be retained in accordance with the Data Protection Act 1998 and all interviews and recruitment processes must be conducted in accordance with UK Equal Opportunities Regulations and Diversity Policies.

In précis, this can be summarised by the **DIVERSITY** mnemonic which provides an all encapsulating definition.

D - Different.

I - Individuals.

V - Valuing.

E - Ethnicity.

R - Religion.

S - Skin, Sex and Sexuality.

I - Intelligence.

T - Talents.

Y - Years.

Also included in this list would be Physical, Physiological and Pedagogical Disabilities.

The objective being to set aside any ignorance or prejudices and be none discriminatory; in effect implementing a level playing field for all candidates and employees through the application of reasonable adjustments. It is sanctioned by Article 14 of the European Human Rights Act 1998 and is implemented through several UK acts of parliament. For more details see the section entitled 'Briefing on UK / EC legislation regarding Employment Law'.

Briefing on UK / EC legislation regarding Employment Law

Equality Act 2010 – replaces previous legalisation denoted with the * prefix

Age Discrimination Act 2006

*Equal Pay Act 1970

*Sex Discrimination Act 1975, 1986

*Race Relations Act 1976

*Disability Discrimination Act 1995

The Human Rights Act 1998

Code of Practice on Racial Equality in Employment 2005

*Employment Equality (Religion or Belief) Regulations 2003

*Employment Equality (Sexual Orientation) Regulations 2003

*Employment Equality (Age) Regulations 2006

Gender Recognition Act 2004

Briefing on UK / EC legislation regarding Recruitment Companies

Employment Agencies Act 1973, 2004

Conduct of Employment Agencies and Employment Businesses Regulations 2003

Briefing on UK / EC legislation regarding Company Law

Health and Safety at Work Act 1974

Workplace (Health, Safety & Welfare) Regulations 1992

Provision & Use of Work Equipment Regulations 1992

Manual Handing Operations Regulations 1992

Health & Safety (Display Screen Equipment) Regulations 1992

Reporting of Injuries, Diseases &Dangerous Occurrences Regulations (RIDDOR) 1995

The Copyright, Design & Patents Act (1988)

The Data Protection Act (1998)

Control of substances hazardous to health **(**COSHH**)** Regulations 2002

Briefing on UK / EC Legislation regards Software Quality Governance

At this time and for the foreseeable future, there is no mandatory UK / EC legal or industry standards required to be adhered to for the testing and documentation of ICT and computer software. So long as the products and services conform to trading standards, contract law and fraud / deception / misrepresentation legislation, then it is permissible to employ any approach and remit deemed appropriate. Indeed if the standard QA mantra of '*we say what we do'*, '*we do what we say'*, '*we prove it*' and '*we improve it*' is followed then the law is duly covered in both spirit and wording. The implementation of which would be achieved through Software Quality Assurance and Software Testing as conducted by suitably qualified and competent professionals who are experienced within the computer software industry; these persons can then act as credible expert witnesses in legal case in accordance with CPR 35 (i.e. the rules and regulations of expert testimony within UK civil cases).

To clarify this point further, in regards to Software Quality Assurance, the most appropriate Act of Parliament would be the **Sale of Goods Act 1979**; which also applies to service provision of tangible items such as computer software. The main three criterion of which being:

- *That the goods and services correspond with the stated description.*
- *That the goods and services be of satisfactory quality.*
- *That the goods and services be fit for the purpose.*

This benchmark has been amended by subsequent legislation; most notably by the **Sale and Supply of Goods Act 1994** which expanded the above definition to include:

- *Wherever goods are bought they must 'conform to contract'.*
- *Goods are of satisfactory quality if they reach the standard that a reasonable person would regard as satisfactory, taking into account the price and any description.*
- *Aspects of quality include fitness for purpose, freedom from minor defects, appearance and finish, durability and safety.*

The full range of UK legislation which is closely coupled to the afore mentioned act include …

- *Consumer Credit Act 1974.*
- *Consumer Protection Act 1987.*
- *Consumer Safety Act 1978.*
- *Contracts (Rights of Third Parties) Act 1999.*
- *EC Unfair Commercial Practices Directive (BERR) 2005.*
- *Fair Trading Act 1973.*
- *Misrepresentations Act 1967*
- *Resale Price Act 1976.*
- *Restrictive Trade Practices Act 1976.*
- *Sale of Goods Act 1979.*
- *Sale and Supply of Goods Act 1994.*
- *Sale and Supply of Goods to Consumers Regulations 2002.*
- *Supply of Goods and Services (Implied Terms) Act 1982.*
- *Trade Descriptions Act 1972.*
- *Unfair Contract Terms Act 1977.*
- *Unfair Terms in Consumer Contracts Regulations 1999.*
- *Unsolicited Goods and Services Act 1975.*

Apart from the *'Hire Purchase Act 1973'* and the *'Weights and Measure Act 1979'*, virtually all aspects of business operations relating to Information Systems are covered by consumer and contract law as listed above.

Although the these acts of legislation are worded for consumer and contract protection, they are also applicable to business-to-business transactions, as under law a 'customer' may be either a person or a business or a government agency.

Traditionally it has often been considered by many software vendors that they are not legally or morally held accountable for the impact of their Software upon their clients; indeed most contracts and licence agreements are worded accordingly with quasi-legal caveats, exemptions and disclaimers. Never the less, in these modern litigious times, the situation is changing rapidly so that instance of successful legal action is becoming more prevalent within both the UK / EC and elsewhere (notably the US). Most cases that are brought to court are undertaken under civil proceedings (corporate and contract litigation) rather than criminal proceedings (trading standards).

Likewise some argue that by offering 'professional services' they can avoid being caught out by the consumer and contract legislation, however, this is a complete fallacy as 'software' products and services are normally 'delivered' and so deemed in legal cases to be 'goods and services'.

In addition to the above laws and those relating to Negligence and Damage inflicted, there is also the Fraud Act 2006 to consider, which represents an entirely new modus operandi of investigating criminal and civil fraud. In that it is no longer necessary to prove a person or organisation has been deceived; this is particularly relevant to software sales!

Fraud Act 2006: Within this new act the focus is now on the dishonest behaviour of the suspect(s) and their intent to make a gain or cause a loss. The new offence replaces the old offences of '*Obtaining property by deception*', '*Obtaining services by deception*', '*Obtaining pecuniary by advantage*', '*Procuring execution of a valuable security*', '*Evading a liability*' and '*Obtaining a money transfer*'. Specifically section 1 introduces a new replacement umbrella offence of '*Fraud*', which can be committed in several ways and is subject to both persons and organisations alike; including …

- *Fraud by false representation (s.2):* Essentially lying about something or someone using any means, including words or actions. An example being promoting goods and services through false representation (e.g. none existent celebrity or professional / public-body endorsements, etc).

- *Fraud by failing to disclose (s.3):* Essentially not saying something when there is a legal duty to do so. An example being failure to declare pertinent information regards goods and services, their suppliers or manufactures which might have bearing upon the business transaction (e.g. obfuscating issues with product suitability and quality, etc).

- *Fraud by abuse of a position of trust (s.4):* Essentially abusing a position of trust where there is an expectation to safeguard the financial and business interests of another person or organisation. An example being false representations of credentials or background (e.g. not declaring a criminal / financial record when asked to do so or presenting false qualifications, etc).

Furthermore, there are several other offences within Fraud Act 2006, all of which, including the above, are indictable. For more detail see the website of the Serious Fraud Office (www.sfo.gov.uk) and the Office of Fair Trading (www.oft.gov.uk).

Indeed it could be argued that the Fraud Act 2006 would cover most instances of pedalling fake, phoney or faulty software! Although the **Sale of Goods Act 1979** still remains the main benchmark in regards Software Quality Assurance. In addition to the Fraud Act 2006 the **Misrepresentation Act 1967** and the **Theft Act 1968 / 1996 (which covers Deception)** also provides further legal enforcement.

As an aside, there is the related issue that regular computer system audits must be undertaken and acted upon, particularly regards the software installed and supplied, as only genuine and legitimate copies of software should be used. Software piracy and theft is a crime which is covered by the Theft Act 1968. See www.fast.org.uk for more details of the Federation against software theft (FAST). Also see www.bsa.org for details of the Business Software Alliance (BSA). For example, software such as

Belarc (http://www.belarc.com/free_download.html) may be used to conduct a generic level audit; it identifies all installed software components on a PC along with their known source. From the resulting report the associated paperwork in terms of contracts and licences can be then inspected to identify any potential discrepancies that need duly investigating. The Belarc software is intended to be installed and used for free on a home PC only, however, for corporate / educational / government organisations a commercial product range is provided and this can be purchased from their website.

***** ADDENDUM #1: PLAN criterion *****

It is necessary to point out that UK legal cases are normally brought or defended on the duel premise of ***DUTY OF CARE*** and ***CONTRACTUAL OBLIGATIONS*** tempered by a test of ***REASONABLENESS***; the standard criteria being ***PLAN*** …

P – Proportionate

L – Legal

A – Accountable

N – Necessary

In the case of software this normally focuses upon multi-user operation, load performance and whether the software actually operates as advertised in accordance with published documentation (including manuals, specifications and test scripts, etc). In law if the client does not like their acquisition but were given suitable 'cool-off' period and full explanation of the product before payment was received then they have little legal remedy unless the product is proven to be faulty or not delivered properly.

***** ADDENDUM #2: Expert Witnesses (CPR 35) *****

Within legal cases it is likely that Expert Witnesses will be employed to give testimony in accordance with CPR 35 (i.e. the rules and regulations governing expert testimony within UK civil cases).

Generally speaking, if any independent Expert Witness was asked to conduct an evaluation of any software on behalf of the plaintiff, then they would probably initially target the multi-user operation and load performance aspects, as these are frequently the most vulnerable weak points in most systems. Likewise next targeted would be those areas of the system where the functionality operates differently (or not at all) in contrary to that advertised or documented and also in contrary to any contractual obligations or promises made as part of the sale; this would include previously working features and functions of the system which have been subsequently removed or stopped operating after an upgrade or database change. In particular they would also seek the 'negative' perspectives and 'negative' experiences of the end-users at the client sites that have reported problems with the software. All this makes excellent testimony for submission within legal cases, particularly if it all the issues are collectively documented and presented as a complete and substantiated package. For these above mentioned aspects are always the easiest to target within most site investigations and provide a readily accessible 'Achilles Heel' which to direct comment, criticism and complaint; especially within legal cases which often result in expensive out of court settlements! Hence it is essential to focus the QA upon these prominent areas of risk within the system and to address any weakness identified as a matter of priority. As the old adage goes '*Forewarned is Forearmed'.*

Indeed the full set of evaluation criteria would likely to include …

Legalities & System Audits …

- *Evaluation of the terms and conditions of contracts, licences and other agreements.*
- *Evaluation of the goods and services delivered compared to that originally advertised and invoiced. This includes associated ROI promised, timesheet and invoice analysis.*

- *Identification of the constituent system components along with confirmation of their nature, actual source, legality, integration and any potential issues arising regarding their usage, specifically looking for any illegally supplied third-party products which have been distributed without royalties being paid and/or legal permission to do so.*

End-user site received training, documentation, acceptance testing and product experience …

- *Evaluation of the user support, training and documentation received from the software vendor including any test databases, case studies / worked examples and UAT scripts provided.*

Non-Functional Operations …

- *Evaluation of multi-user operation (WAN / LAN) with user access and system feature privileges; including evaluation of the record locking mechanisms employed and also establishing the actual 'maximum' number of concurrent users supported by the system compared to that advertised or promised.*

- *Evaluation of database volume generation, including any system capacity gauging provided by the software vendor or that which can be conducted by the end-user site. The concept being to establish appropriate database volumes that correspond to likely future usage predicted for the client site.*

- *Evaluation of system load & performance, particularly within large scale network and database environments.*

Usability of the user interface (UI) of the software …

- *Evaluation of menu & function navigation, input / edit field validation and operation of screen objects.*

- *Evaluation of the UI in terms of system intuitiveness, consistency and conformity to stated formats, industry standards and 'house styles'.*

Functionality …

- *Evaluation of the end-user site operation including difficulties experienced by users and assessment of the business contingency procedures provided.*

- *Evaluation of the functionality of the software and its multi-user performance against the vendors' own specification / user / training documentation; essentially establishing whether the software operates as advertised and as per contractual obligations.*

- *Evaluation of the underlying system operation focusing upon numerical accuracy, functionality and stability using industry standard QA techniques and methodologies such as those defined by BS7925. Including conducting evaluations of UAT scripts provided by the software vendor for use by client sites.*

Software vendor …

- *Benchmark of the professionalism and credentials of the vendor by evaluating against the 'BCS Code of Conduct' and the 'BCS Code of Practice'.*

Briefing on Justification for Software Quality Assurance

This is a very good point of issue to raise which is best answered by the following question...

'How would you feel, if the expensive accounts software you purchased, and that your business heavily relies upon, showed any or all of the following symptoms?'

- Can't add up properly and experiences rounding errors.
- Posts financial transactions to the wrong client account.
- Produces reports and enquiries that resemble works of fiction.
- Slow and cumbersome in operation whilst prone to system crashing or freezing.
- Draconian single user system operation being passed off as multi-user software.
- Likewise no locking employed so several concurrent users can overwrite each other's work.
- Data stored on file cannot be retrieved in its entirety or analysed appropriately.
- Critical data frequently corrupted, duplicated or destroyed.
- And so on …

Regrettably these symptoms are still far too prevalent in many IT systems; the reason simply being a lack of appropriate Software Quality Assurance! This would include the associated Technical Documentation and Software Testing; indeed it is very true to say that Software Quality Assurance has a serious and long standing image problem to address.

Unfortunately for many years, such an important activity has regrettably gained an unenthusiastic reputation by many within the Computer Industry; particularly those who are responsible for software development and sales. As a result Software Quality Assurance is often only conducted half-heartedly and usually as an afterthought or just simply as a causal formality; all too often it is the clients and their end-users who find out the actual truth too late and often the expensive way. This is very reminiscent of the preverbal three monkeys; hear no evil, see no evil and say no evil. The evil in this case being the software functionality not being as advertised or specified. Likewise with multi-user operation, user interface operation, load performance and stability issues.

Sadly it is still claimed by many that Software Quality Assurance is far too costly and troublesome; particularly in terms of time, manpower and cost required. Worse still is that this important responsibility is often expected to be undertaken by persons who often do not have the necessary authority, or the skills-set, or the resources, or even the ethics for this task. Furthermore when genuine concerns and feedback are raised they are all too often dismissed, ignored or simply derided in a patronising and aggressive manner.

Regrettably, the impact of this negativity and apathy can be far reaching in terms of its consequences. In these modern days technology is heavily relied upon, rarely questioned and indeed commerce would not survive without it. Yet people rarely question why issues arise and who is responsible.

Quis custodiet ipsos custodes*?*

Briefing on QA Plan (based upon IEEE 829 implementation)

Employing a generic QA PLAN provides for cost-effective and efficient IEEE 829 implementation without the need for extensive volumes of associated test documentation; thus allowing for only the essential aspects to be focused upon within the test scripts and test results.

QA PLAN (TESTING BENCHMARK)

Primary Framework Employed: IEEE 829 – 1998 Standard for Software Test Structure

- *4.2.1 Test plan identifier;*
- *4.2.2 Introduction;*
- *4.2.3 Test items;*
- *4.2.4 Features to be tested;*
- *4.2.5 Features not to be tested;*
- *4.2.6 Approach;*
- *4.2.7 Item pass / fail criteria;*
- *4.2.8 Suspension criteria and resumption requirements;*
- *4.2.9 Test deliverables;*
- *4.2.10 Testing tasks / procedures;*
- *4.2.11 Environmental needs;*
- *4.2.12 Responsibilities;*
- *4.2.13 Staffing and training needs;*
- *4.2.14 Schedule (inc. Estimated Timings);*
- *4.2.15 Risks and contingencies;*
- *4.2.16 Approvals.*

Primary Methodology Employed: BS 7925 Standard for Software Unit Test Methods

4.2.1 Test plan identifier …

[Software Name]

4.2.2 Introduction …

The purpose of the QA Plan is to implement the following QA Mission Statement and Approach…

TO PROVIDE THE ORGANISATION WITH A DETAILED SPECIFICATION OF THE TYPE, DURATION, PERFORMANCE MEASURES AND ASSOCIATED COSTS OF TESTING THE [Software Name] AND APPLYING "FIT FOR PURPOSE" QUALITY ASSURANCE FUNCTIONS.

The UK legislation used as the primary QA benchmark being the Sale of Goods Act 1979; see the appendix section for more details.

4.2.3 Test items …

All items are to be tested according to their designated test coverage levels as determined by the QA Manager and mandated by the Management Team at ***[Organisation Name]***.

4.2.4 Features to be tested …

All existing features documented within the User Manuals, Technical References and QA / Test Strategies are to be tested to the level prescribed as determined by the QA Manager and mandated by the Management Team at ***[Organisation Name].***

All new and amended features are to be initially tested in accordance with the requests / instructions issued by the Developers and Management; this should ideally be supplemented by the provision of a working demonstration of the product which is to be accompanied by an technical explanation of the new / changed features along with recommended testing strategies to be followed and a précis of the testing already conducted.

4.2.5 Features not to be tested …

All features outside of the User Manuals, Technical References and QA / Test Strategies are NOT to be tested; however, where such features are found and considered suitable / necessary for inclusion then these need to be reported to the QA Manager who would determine the appropriate course of action including the sanctioning of any additional testing and / or production of new test scripts.

4.2.6 Approach …

A three level hierarchical approach is employed for test coverage:

- First level of coverage focuses upon the user interface which is tested against the User Manuals. Predominately exploring the menu navigation, dialogue design and system options available to the user.

- Second level of coverage focuses upon breadth of testing rather than depth of testing; in which functions are tested at an initial high level of detail predominately constituted by Technical Reference Documentation and associated QA / Testing Strategies. The Technical Reference Documentation is an extension to the User Manuals and may contain excerpts from the Design Specifications which are approved for public distribution. The QA / Testing Strategies are constituted by one or several paragraphs of narrative scripts that describe in summary the method of testing to be followed. The QA / Testing Strategies may be additionally supported by mathematical spreadsheet simulations, lifecycle diagrams and test scripts (constituted by summary descriptions of test-setup, test-methods, test-procedures, test-inputs and expected-outcomes). The techniques and tools employed are as per BS 7925 which define the stated methods for unit testing.

- Third level of coverage focuses upon depth rather than breadth of testing and is targeted upon specific core functions, including accounting and scheduling, which are required to be tested at low level focusing upon the detailed results and underlying logic. This follows a formalised approach of test-setup, test-methods, test-procedures, test-inputs and expected-outcomes which are specified accordingly at detail. The techniques and tools employed are as per BS 7925 which define the stated methods for unit testing.

Additional testing would also include Load Performance and Multi-User Networking. Likewise with testing of Installs, Updates and Data Conversions, etc.

4.2.7 Item pass / fail criteria …

As determined by the QA Manager given the circumstances at the time and the instructions received from the Management Team at ***[Organisation Name]***; all issues found are recorded within the in-house Issue tracking system for on-going resolution monitoring and QA Metrics statistical analysis.

All new and existing issues of prominence or concern are discussed with the Management Team at ***[Organisation Name]***; these issues are then formally prioritised and tasked accordingly. Appropriate action includes priority escalation and de-escalation, as well as, conducting a further investigation into the report issues.

4.2.8 Suspension criteria and resumption requirements …

As determined by the QA Manager given the circumstances at the time and the instructions received from the Management Team at ***[Organisation Name]***; all issues found are recorded within the in-house Issue tracking system for on-going resolution monitoring and QA Metrics statistical analysis.

All new and existing issues of prominence or concern are discussed with the Management Team at ***[Organisation Name]***; these issues are then formally prioritised and tasked accordingly. Appropriate action includes priority escalation and de-escalation, as well as, conducting a further investigation into the report issues.

4.2.9 Test deliverables ...

An evaluation of the current version stability and functionality from which the decision to release to client sites can be determined accordingly based upon substantiated evidence.

NB: It is the Management Team at ***[Organisation Name]*** who makes the finale decision to release software to client sites based upon the recommendations of the QA Manager.

4.2.10 Testing tasks / procedures ...

Refer to the following constituent documents:

- User Manuals.
- Technical Reference Documentation.
- QA / Testing Strategies.
- Mathematical Spreadsheet Simulations.
- Lifecycle Diagrams.
- Test Scripts; incorporating …
  test-setup, test-methods, test-procedures, test-inputs and expected-outcomes.

4.2.11 Environmental needs ...

Standard ***[Software Name]*** installation for the current program version with suitable database provided. Variations would include stand-alone and network configurations with various permutations of operating systems, databases, browsers and third party software, etc.

4.2.12 Responsibilities ...

The QA Manager is responsible for all aspects of Software Testing and Quality Control.

It is the responsibility of the QA Manager to inspect and approve ALL testing which is to be presented to the Management Team at ***[Organisation Name]***, or which impacts upon a software release to client sites, or which is to be published / distributed externally by the company.

NB: The Management Team at ***[Organisation Name]*** include representatives of the QA, Development, Support, Sales and Training teams, as well as, Company Board Directors.

4.2.13 Staffing and training needs ...

Flexible in accordance with the circumstances at the time.

Everyone is encouraged to participate and contribute to the product testing and company QA function; to this end a company library is maintained and available to all. Likewise guidance and training on such matters is available from the QA Manager.

NB: The QA Manager is a suitably qualified and experienced expert in the field of Quality Assurance and Software Testing; the official remit for whom is to implement the QA Mission Statements (as detailed in 4.2.2).

4.2.14 Schedule …

Flexible in accordance with the circumstances at the time including budgets, deadlines and manpower resources; however the prescribed testing cycle is ideally ***[specify number of weeks; e.g. 4]*** weeks to account for testing of new functions, retesting of existing functions, compilation / maintenance of documentation / test-scripts and updating the progress of issues from which statistical analysis of QA Metrics can be generated. These tasks are not conducted one task per ***[specify periodical; e.g. week]*** covering a single ***[specify number of weeks; e.g. 4]*** week test cycle but collectively within ***[specify number of cycles; e.g. 4]*** mini test cycles; each of which being of ***[specify number of weeks; e.g. 1]*** week's duration where by all testing is conducted within a repetitive multi-tasking environment. Essentially following an AGILE / Scrum approach where by testing / retesting is conducted iteratively and incrementally focusing upon releases having phased deliveries of functionality whose size and scale is readily manageable. On occasion linear progression within the ***[specify periodical; e.g. weekly]*** testing cycles is required (i.e. not employing a multi-tasking approach) and this will be authorised by the QA Manager.

It is estimated that to exercise all aspects of the User Interface, Functionality, Multi-User and Performance using the test scripts provided requires a MINIMUM of at least a ***[specify number of weeks; e.g. 6]*** weeks; conducted full time and uninterrupted with the *assumption* that there will be no new issues found or new test scripts required to be created!

4.2.15 Risks and contingencies …

On-going dynamic risk assessments are conducted within action / contingency plans being implemented accordingly to mitigate potential impact; the documentation for which is deemed not required for mandatory retention however when necessary issues are escalated for the attention of the Management Team and these are raised within the ***[specify periodical; e.g. weekly]*** QA Meetings.

4.2.16 Approvals …

In the first instance all formal testing conducted is inspected and approved by the QA Manager and this is then subsequently presented for ratification by the Management Team at ***[Organisation Name].***

QA Metrics statistical analysis is presented periodically to the Management Team at ***[Organisation Name]*** during the QA Meetings; thus providing an evaluation as to the state of the current version of the software in terms of prioritised outstanding issues and indicating its suitability for release. It is the stated company objective to ensure that software releases to customer sites does not contain known Priority ONE or Priority TWO issues.

- Priority ONE – Major Issue with no work-around requiring priority attention.
- Priority TWO – Major Issue with work-around.
- Priority THREE – Minor Issue.
- Priority FOUR – Lesser Priority THREE issue.
- SUGGESTION – Proposal for new Feature; prioritised using MoSCoW mnemonic … MUST / SHOULD / COULD / WONT

NB: Once prioritised then each issue is assigned a progression STATUS; the classification for which is either 'Decision Required', 'Rejected', 'Pending', 'In-Progress' or 'Completed'.

Where agreed by the Management Team to do so, both Priority ONE and Priority TWO issues may be downgraded or subject to a moratorium as appropriate. Typically this would be

undertaken for products subject to a substantive rewrite, expansion or winding down in preparation for being phased out of production.

Compiled By & Date …

[QA Manager Name]

[Organisation Name]

QA Manager

[Date of Document]

Appendix: UK Legislation (Sale of Goods Act 1979) …

This legislation is used as our primary benchmark for Software Testing and QA; the main three criterion of which being:

- That the goods and services correspond with the stated description.
- That the goods and services be of satisfactory quality.
- That the goods and services be fit for the purpose.

This Act of Parliament has been amended by subsequent legislation; most notably by the Sale and Supply of Goods Act 1994 which expanded the above definition to include:

- Wherever goods are bought they must 'conform to contract'.
- Goods are of satisfactory quality if they reach the standard that a reasonable person would regard as satisfactory, taking into account the price and any description.
- Aspects of quality include fitness for purpose, freedom from minor defects, appearance and finish, durability and safety.

Other Acts of Parliament also apply, including those covering trading standards, contract law and criminal law; however, in regards to Software Quality Assurance, the most appropriate Act of Parliament would be the above. Indeed this Act of Parliament also applies to service provision of tangible items such as computer software, training and installation / support.

End of QA Plan

Briefing on Preparing for the Post Implementation

Although it is very true to say that buying software 'off-the-shelf' is far less risky for the client than commissioning bespoke software, there is always the usual problems to be accounted for within the implementation cost and timescales. These should be made very clear to clients as part of the customer relations and pricing policy and they should be encouraged to implement appropriate risk mitigation strategies and account for them accordingly within their project plans ...

- It is critical to win people's 'hearts' & 'minds'. Users in general do not like change, particularly if they disagree with it or cannot see the point or are regularly confronted by error messages from the new or upgraded system. Regardless of the justification for the new implementation, they will resist and even derail any new projects unless they are fully reassured, brought onboard and encouraged to fully cooperate. It is necessary that both the software vendor and the client appoint their own 'champions' to oversee the introduction and ongoing usage of the software. Likewise the intended design of the system should be open, accountable and transparent so that its operation can be readily interpreted and applied accordingly by the clients and their end-users.

- Users have to be trained in the use of the new software along with its associate protocols, procedures and documentation. They also need the necessary practice in applying their new skills. Furthermore the likelihood is that mistakes will be made and this should be factored. Hence a substantial 'tool-up' and 'training' period is required between 'installation' and 'going-live'.

- Extensive functional, integration and system testing has to be undertaken prior to going 'live' by BOTH the software vendor and the client. This operational 'shake-down' is necessary to ensure the new and upgraded software operates in accordance with the specified requirements and to accepted standards. All too often 'horror stories' abound where organisations have paid dearly for the mistakes of their software vendors and of their in-house IT teams. It should be appreciated that client organisations also have their own customers and regulatory bodies to whom they are held accountable.

- Customisation may be required to adapt the software for use within the client site, however, this will be expensive (particularly for any software changes and associated testing). Instead it is better to encourage the client site to adapt to the software. Likewise it advisable to sound out potential enhancements with clients and staff before development is sanctioned; where appropriate implementing these directly within the main product line well in advance of the due deadline date, however, such enhancements must be configured for exclusion by all the other clients until ready for official release.

- Where possible avoid data conversions, either from previous versions or other products, as they can be seriously problematic and costly. Where this is not possible then this should be realistically accounted for within the costing and timescales. In particular it is necessary to factor for any expensive re-work that might be required where data-conversions have not kept pace with product development.

- The clients must declare exactly their 'wants' and 'needs' in terms of the system operation from the onset and always before any changes / additions are implemented. Otherwise it could be a case of consultants ad infinitum with extortionate fees to match!

- Implementation will never stop, as there will always be 'modifications' and 'new additions' to be included. Not just new releases of the software to be installed but also additional client instigated customisation. Software is a long term commitment, particularly where the clients are dependent upon it for their business being a success.

- Post-Implementation Blues! Any new software will wreak havoc and the Return on Investment (ROI) is normally very long-term. Clients should be prepared for an initial rough ride and a steep learning curve; however, this will smoothen-out and the system will become intuitive over time. Although proper installation, configuration and training will significantly minimise the impact experienced.

Briefing on Employing Outsourcing & Subcontracting

Raison d'être & ROI …

The justification for considering offshore outsourcing and subcontracting is that it is promoted as being cheaper, quicker and easier to conduct for projects that have been standardised to conform to the criterion of being specific, measurable, attainable, relevant and time-bound. This is particularly the case within software testing and its associated quality assurance (QA). Being known entities, they can in theory be quantified and qualified; hence, they can be readily packaged and project managed once the necessary test scripts and technical documentation has been compiled.

The realities being that the price differential does significantly and clearly speaks for itself. Exact costs can only be achieved upon application with the Offshore Outsourcers and UK Subcontractors as pricing does fluctuate, however, Offshore Outsourcing and UK Subcontracting are progressively becoming more popular for very good financial and productivity reasons.

The practical upshot is that within software testing and its associated QA, it is the case that the more technical documentation and test scripts that are packaged at the onset, then the more process workers that can be employed by Offshore Outsourcers and UK Subcontractors to undertake the same task with the minimum of supervision required. As the saying goes 'Many hands make light work!'

Furthermore this concept of using process workers for repetitive testing of 'signed-off' program versions and test scripts is both cost efficient and extremely effective. Obviously it is necessary to specify the level of competency and skills-set required for process workers and these could be classed as Experienced IT Professionals, Graduates and Non-Graduates. Indeed the extensive use of non-graduates is both feasible and viable in that compared with graduates and experienced professionals, it is the case that non-graduates are more likely to emulate the behaviour and mind-set of the client's end-users in regards to any software issues arising. Indeed, these days most non-graduates are themselves are highly educated with degree equivalent in other non-IT disciplines whilst being extremely computer savvy.

Throughout the process workers are supervised by a management structure that is staffed by qualified professionals who are specialists in the QA field. When the necessary technical documentation and test scripts are not available, then their compilation and review can be readily compiled through UK Subcontractors. Alternatively suitably qualified Offshore Outsourcers can be instructed to undertake this task. Likewise UK Subcontractors can be tasked to conduct 'Repetitive' testing such as Software Trials and Regression Testing.

The offshore outsourcing and subcontracting is normally conducted in fixed term renewable contracts, which are priced according to the specified deadline and manpower resources (i.e. number of participants, their skill-level and their periodical participation throughout the contracted period). All work commenced is usually paid for in advance; however, the monetary values involved can be kept affordable through employing frequent short-term review points where the contract can be continued, terminated, upsized or downsized as required within the terms of the contract.

Terminology …

- QA - Quality Assurance
- R&D - Research & Development

Benefits of Offshore Outsourcing for Software QA (as compared Software R&D) ...

1. Software testing and its associated QA are by nature known entities that can be quantified and qualified accordingly; hence they can be readily packaged and project managed.

2. Client intellectual property rights, confidentiality and copyright are protected within the contract agreements; with access to source code and unauthorised materials being prohibited

accordingly. Hence significantly mitigating the associated risk involved in regards third party exposure.

3. Those aspects of the software testing that are repetitive in nature may be successfully offshore outsourced with significant cost savings and improved efficiencies. Once a test script is compiled and signed-off for a particular program version, it can then be despatched for external testing on an on-going basis to ensure continuity of operation.

4. Repetitive testing is an ideal candidate for cost-efficiency savings as the work involved can be undertaken by process workers with minimal supervision; the prerequisite is that the necessary Mandatory Materials are provided at the onset.

Mandatory Entry Requirement …

1. Instructions & Variations are signed-off by the client and agent.

2. Financial payment is normally required in advance and at regular periodic intervals; accompanied by specified work, manpower requirements and timescales.

Mandatory Materials …

1. Install Disks and/or Internet Download and/or Remote On-Line Access (e.g. VPN). Only the executable versions of the software are actually required! Revised versions of the executable programs as released by the Software R&D team would have to be forwarded to the Offshore Outsourcers and UK Subcontractors upon issue.

2. Technical Specs, Staff Manuals, User Manuals and Test Scripts plus any other technical documentation available.

3. Known Issues.

Mandatory Deliverables Returned …

1. Issues Found and Progress Reports.

Mandatory Exit Criteria ...

1. Contracts may be continued, terminated, modified, upsized or downsized at the agreed interim milestones in accordance with the contract terms.

Optional Extras ...

1. It is appreciated that due to commercial constraints, the necessary Mandatory Materials may not always be available within the in-house QA; however, often these can be readily compiled through UK Subcontractors and Offshore Outsourcers.

Implementation Summary ...

1. Activities specific to Software Testing and its associated QA.

2. Focused upon ‘Repetitive’ testing; Software Trials and Regression Testing.

3. Two work streams; both concurrent and on-going in their implementation.

    a. Software Trials are commenced at the preliminary stages and deliver immediate feedback regarding stability and robustness in operation.

    b. Regression Testing is commenced at the interim and later stages; delivering responsive feedback in regards continued functional compliance.

4. Both Software Trials and Regression Testing can be undertaken for single or multiple projects which are normally long-term and may spawn several sub-projects.

5. Software Trials are conducted from the perspective of the end-user and in accordance with the supplied installation guides and system manuals; covering …

    a. Software installation / updates / upgrades.

    b. Data conversions.

    c. Third-party software integration.

    d. Record-locking operation.

    e. Load performance.

    f. Single-User (Standalone PC) and Multi-User (Networked PC) configurations.

    g. Exercising of advertised / documented functionality and assessment of user interface.

    h. Permutations of operating system, web browser and database evaluated.

6. Regression Testing is conducted in accordance with the test scripts provided; these would have previously signed-off by the in-house QA team against specific program versions and cost estimated in terms of manpower, timescales and practicalities required to complete.

7. Offshore Outsourcers normally conduct the Software Trials and Regression Testing with UK Subcontractors compiling any additional test script and technical documentation. However, where necessary, Offshore Outsourcers can also compile additional test scripts and technical documentation. Likewise UK Subcontractors can also conduct the Software Trials and Regression Testing where necessary.

QA Project Scoping Evaluation Matrices …

Within Appendix there are three document templates;

1. Appendix A - Project Scoping & Mandatory Materials Matrix

    Project Scoping & Mandatory Materials Matrix is used to conduct initial investigations and specifications in which the presence of the mandatory materials required for software testing is evaluated accordingly. This provides a vehicle to scope the various aspects involved within the QA project work, which then can be presented to Offshore Outsourcers and UK Subcontractors so that they can compile their cost estimates and bid tenders.

2. Appendix B - Project Staff Requirements, Timescales & Budgets Matrix

    Project Staff Requirements, Timescales & Budgets Matrix is intended to record supplementary information that specifies the manpower resources sought for the software testing and its associated QA. It is not mandatory for this additional matrix to be completed; however, it is strongly recommended that it is compiled in full as it would significantly assist Offshore Outsourcers and UK subcontractors in compiling their cost estimates and bid tenders. Most of this information would be sourced from prior in-house QA testing and so should be readily available.

3. Appendix C - Project Statistics Matrix

    This contains the minimum and maximum boundaries for budgets and timescales. It is an important factor to remember that financial payments are normally required in advance and at

regular periodic intervals in accordance with contract terms. This information provides Offshore Outsourcers and UK Subcontractors with extremity limits for the project timescales and budgets.

CAVEAT: The very minimum materials required for Software Trials are the Install Disks / Internet Download / Remote On-Line Access and the System Manuals. In the case of Regression Testing, the Test Scripts are also required. If these are NOT present then testing cannot commence; indeed serious questions should be asked as to WHY they have not been conducted.

Project Scoping & Mandatory Materials Matrix …

Throughout this scoping exercise, the prime objective of the Evaluation Matrix is to fully develop the available mandatory materials required so that the construction and review of software test scripts and technical documentation can be successfully undertaken by both the in-house QA team and external third parties alike. The mandatory materials required are as follows:

1. Project Mandate and Schedules.
2. Technical Specs.
3. User Manuals.
4. Staff Manuals.
5. Test Scripts.
6. Install Disks / Internet Download / Remote On-Line Access; i.e. including associated Installation Guides.
7. Known Issues.
8. Software Trials.
9. Regression Testing.
10. User Interface (UI) - Menus & Dialogues.
11. Functionality as Advertised / Described.
12. Envisaged System Configurations (i.e. environment options including operating systems, databases, internet browsers, third-party software and network configurations, etc.)
13. Non-Functional & Other (i.e. Multi-User Operation, Load Performance Operation, Website and Database Volumes, etc.)

NB: System Manuals are constituted from the Technical Specs, User Manuals and Staff Manuals.

The five columns are in respect to the Size & Complexity Gauging based on the paperwork available;

1. Number of Documents Available
2. Total A4 Sheets
3. Approx. Timings (per single sheet for a single system configuration)
4. Number of Samples Provided
5. Content Assessment (Accurate? / Complete? / Current?).

They are all used to cost estimate the work involved (e.g. manpower requirements, budgets and schedules, etc.). They also provide a representative sample of the system testing which can be provided to UK Contractors and Offshore Outsourcers for their consideration as part of the project tendering process.

NB: The full set of Mandatory Materials should NOT be externally distributed until an agreement with all parties involved has been signed off and the legal aspects covered, however, it should be ready and available for escalation and immediate commencement.

Project Staff Requirements & Timescales Matrix …

Additional manpower requirements matrix would be constituted from the following roles;

1. Outsourcing Agent (UK based)
2. Experienced IT Professional (Offshore General Manager)
3. Experienced IT Professional (Offshore Line Manager)
4. Experienced IT Professional (Offshore Software Trials)
5. Experienced IT Professional (Offshore Regression Testing)
6. IT Graduate (Offshore Software Trials)
7. IT Graduate (Offshore Regression Testing)
8. Non-Graduate (Offshore Software Trials)
9. Non-Graduate (Offshore Regression Testing)
10. Other Staff: Legal Advisor
11. Other Staff: Financial Advisor
12. Other Staff: Technical Advisor
13. UK Contractor: Software Tester / QA Analyst

Obviously not all of the above roles will be required and indeed the same person could perform several, however, demarcation of roles is required if only to ascertain fractional components in terms of allocated participation.

For each role the necessary attributes and allocations would be assigned;

1. Staff Grading (General Manager, Line Manager, Software Tester, Other and Contractor)
2. Quantity Required
3. Contribution Required (Quarterly / Monthly / Weekly / Daily / Ad-Hoc)
4. Time Allocation Required (as per Contraction Required)
5. Remarks

Project Statistics Matrix …

In addition several project statistics would be required from the onset in relation to boundary limits for interim milestones and payments;

1. Minimum Project Commencement Date
2. Maximum Project Commencement Date
3. Minimum Project Completion Date
4. Maximum Project Completion Date
5. Minimum Number of Weeks of Work (Approved)
6. Maximum Number of Weeks of Work (Approved)
7. Minimum Projected Budget (if known)
8. Maximum Projected Budget (if known)
9. Minimum Periodic Interval (Quarterly / Monthly / Weekly / Daily / Ad-Hoc)
10. Maximum Periodic Interval (Quarterly / Monthly / Weekly / Daily / Ad-Hoc)

Finally a necessary reality check...

In theory preparing for outsourcing is simply an extension of the QA Plan and in-house Software Testing activities; in essence it is just expanding the resource allocations for manpower, budgets and workload so that QA can also be conducted externally in addition to being conducted in-house. However, CAVEAT EMPTOR … Let the buyer beware! In reality both Offshore Outsourcing and UK Subcontracting are not magic bullet solutions. Furthermore in-house QA should NEVER be discontinued (or downsized) as essential product knowledge and inspectorate / standards-enforcement would be permanently lost from within the organisation.

The recipe for good Offshore Outsourcing and UK Subcontracting success requires engaging the services of a reputable UK based Agent with a proven track record who can be contracted to do all the hard work and who is legally and financially held accountable.

Contrary to most misconceptions, Offshore Outsourcing and UK Subcontracting is neither cheap, nor quick, nor easy, nor is any success guaranteed at the end of the project; but Offshore Outsourcing and UK Subcontracting can be extremely cost effective and efficient when significant volumes of standardised and none critical work needs to be conducted over a long period of time with no resources available in-house to undertake the task. The work can either be undertaken offshore by Offshore Outsourcers in countries such as China and India and / or by Subcontractors here within the UK. It is strongly recommended that you employ a reputable and established Offshore Outsourcer and / or UK Subcontractor whose Agent is based in the UK and who can be readily contacted and who is prepared to conduct regular 'face-to-face' progress meetings and deal with any matters arising in a timely and professional fashion. Ideally the Agent should be able to offer appropriate securities and guarantees for all work conducted. It is also necessary that the work is packaged accordingly so that it can be both 'conducted piecemeal' and 'aborted abruptly' at any time in a safe and systematic manner as part of an official exit strategy. Never diverge any company sensitive material or intellectual property (IP); instead these should be kept securely at the highest level within the organisation. Any work that is outsourced or subcontracted must first be compartmentalised in to separate discrete components so that they can be distributed accordingly across different Offshore Outsourcers and / or UK Subcontractors. None of the Offshore Outsources and / or UK Subcontractors should ever have knowledge of each other and their assigned system component(s) to test should not provide them with any insight into the full system design (or indeed any significant part of it). This may appear rather clandestine and Machiavellian but it does allow full secrecy and control to be maintained in regards valuable products and components. Regrettably software theft and plagiarism is extremely easy these days and it is readily achieved once their internal mechanisms have been either publicly disclosed or the internal workings determined.

Furthermore it is no good simply ‘Googling’ for Offshore Outsourcers and / or UK Subcontractors hoping for the best; although attractive pricing can be readily sourced and golden promises are often made to secure a deal. The problem is guaranteeing that the end product will in fact be delivered correctly and consistently, as well as, being on time, to specification and within budgets. You should NOT let yourself be in a position of becoming ‘tied to contract’ or ‘dependent’ otherwise undue liberties and malpractice may ensue. Legal redress for which would be protracted, costly and with no guarantees of success. Likewise you should ensure your business operations are NOT impacted nor compromised as a result of the Offshore Outsourcing and UK Subcontracting process or its failure.

Finally, there is the slight matter of payment, in that many Offshore Outsourcers and UK Subcontractors require a retainer upfront and regular payment throughout; although this can be negotiated it is still likely that the actual costs involved will rise substantially and deadlines will be busted regularly unless proper project management is established and maintained throughout. Hence the need for having suitable specifications upfront that are properly scoped and agreed upon by all parties in advance to include the associated budgets / deadlines / deliverables; likewise the nature of the work to be undertaken should be conducive to Offshore Outsourcing and UK Subcontracting (for example such as Software Testing and its QA which can be standardised and packaged accordingly). Thus to reiterate the earlier recommendation of engaging the services of a reputable UK based Agent with a proven track record who can be contracted to do all the hard work and who is legally and financially held accountable.

Appendix – A: Project Scoping & Mandatory Materials Matrix …

Mandatory Materials	No. of Documents	Total A4 Sheets	Approx. Timings	Content Assessment
Project Mandate & Schedules.				
Technical Specs.				
User Manuals.				
Staff Manuals.				
Test Scripts.				
Install Disks.				
Known Issues.				
Software Trials.				
Regression Testing.				
UI - Menus & Dialogues.				
Functionality as Advert / Described.				
System Configurations.				
Non-Functional & Other.				

Project Scoping & Mandatory Materials Matrix is used to conduct initial investigations and specifications in which the presence of the mandatory materials required for software testing is evaluated accordingly. This provides a vehicle to scope the various aspects involved within the QA project work, which then can be presented to Offshore Outsourcers and UK Subcontractors so that they can compile their cost estimates and bid tenders.

Appendix – B: Project Staff Requirements & Timescales Matrix …

Staff Grading	Quantity Required	Contribution Required	Time Allocation	Remarks
Outsourcing Agent (UK based).				
Experienced IT Professional (Offshore General Manager).				
Experienced IT Professional (Offshore Line Manager).				
Experienced IT Professional (Offshore Software Trials).				
Experienced IT Professional (Offshore Regression Testing).				
IT Graduate (Offshore Software Trials).				
IT Graduate (Offshore Regression Testing).				
Non-Graduate (Offshore Software Trials).				
Non-Graduate. (Offshore Regression Testing).				
Other Staff: Legal Advisor.				
Other Staff: Financial Advisor.				
Other Staff: Technical Advisor.				
UK Contractor: Software Tester / QA Analyst				

Project Staff Requirements, Timescales & Budgets Matrix is intended to record supplementary information that specifies the manpower resources sought for the software testing and its associated QA. It is not mandatory for this additional matrix to be completed; however, it is strongly recommended that it is compiled in full as it would significantly assist Offshore Outsourcers and UK Subcontractors in compiling their cost estimates and bid tenders. Most of this information would be sourced from prior in-house QA testing and so should be readily available.

Appendix – C: Project Statistics Matrix …

Minimum Project Commencement Date	
Maximum Project Commencement Date	
Minimum Project Completion Date	
Maximum Project Completion Date	
Minimum Number of Weeks of Work (Approved)	
Maximum Number of Weeks of Work (Approved)	
Minimum Projected Budget (if known)	
Maximum Projected Budget (if known)	
Minimum Periodic Interval (Quarterly / Monthly / Weekly / Daily / Ad-Hoc)	
Maximum Periodic Interval (Quarterly / Monthly / Weekly / Daily / Ad-Hoc)	

This contains the minimum and maximum boundaries for budgets and timescales. It is an important factor to remember that financial payments are normally required in advance and at regular periodic intervals in accordance with contract terms. This information provides Offshore Outsourcers and UK Subcontractors with extremity limits for the project timescales and budgets.

Briefing on Compiling Reverse Engineered Specifications and Technical Guides

Primarily this would be undertaken by QA for in-house use (namely the 'positive' or +ve functional testing which is intended to prove that the product operates as described / marketed) and in particular used as the roadmap for compiling various Test Catalogues and Functional Test Scripts; with the intention of disseminating the system design and new features to colleagues in support, training and sales/marketing. Normally clients and other third parties would only be sent documentation from the training and sales / marketing departments with the content having been first officially approved for external communications.

The approach advocated is to form a pictorial 'story board' of screenshots with suitable annotations that depict the end to end operation of the system (i.e. full lifecycle) that can then be followed through page by page; in particular the various variations in the system usage would be included, however, the standard options would be portrayed with the most emphasis throughout. The easiest approach being one screenshot per page with bullet point annotations to provide explanatory notes.

However, care must always be taken to ensure that the contents is neither company sensitive, nor potentially offensive and must never contain any of the clients details or their data; the reason being that QA documents are often distributed externally and may not be properly vetted further along the line. Hence it is always best to compile QA documents as if they will be externally distributed.

This concept of the roadmap is very important as the quality and quantity of the Reverse Engineered Specifications and Technical Guides should permit self-training to be conducted without actually having to use the product first-hand; encouraging staff and clients to train themselves and make their own product evaluations. Likewise with functional testing, as the roadmap encourages the compilation of procedures that are repeatable regardless of who undertakes the testing and how often it is repeated or when it is repeated.

All pertinent information should be included within the annotations including inputs, outputs, caveats and calculations/wizards so the system operation is understandable and relevant in regards to its intended usage. Where appropriate, the mathematics and data processing should be modelled using spreadsheets; hence they can be used by both QA and Training to check the results.

As old adage goes '*A picture paints a thousand words'*.

It is true that there are no definitive or legally mandated obligations in regards Software Requirements Specification, however, IEEE 830-1998 does provide an established industry standard that presents appropriate best practices that can be readily followed and which would be acceptable within most Information Technology projects. It can either cover the whole system operation or be specific to new functionality. The above technique of Compiling Reverse Engineered Specifications and Technical Guides provides a starting point and roadmap which can then be further defined and refined using an iterative but incremental approach; where upon the resultant documentation can be duly expanded to comply with the full set of criteria for IEEE 830-1998.

Briefing on IEEE 830-1998: Software Requirements Specifications (SRS)

There are no definitive or legally mandated obligations in regards Software Requirements Specification, however, IEEE 830-1998 does provide an established industry standard that presents appropriate best practices that can be readily followed and which would be acceptable within most Information Technology projects. It can either cover the whole system operation or be specific to new functionality. However, as a practical example, a standard template has been provided within the section on Sample Quality Management System > Sample QA Documentation > User Requirements (pages 29 & 30). Regardless of the method employed for the SRS; to the customers, suppliers and other third parties a good SRS should provide several specific benefits, such as the following …

- Establish the basis for agreement between the customers and the suppliers on what the software product is to do.
- Reduce the development effort and risk involved.
- Provide a basis for estimating costs and schedules.
- Provide a baseline for validation and verification.
- Facilitate transfer from supplier to client and third-parties.
- Serve as a basis for future enhancement.

Criterion for producing a good SRS specified in IEEE 830 …

- **Nature of the SRS;** including functionality, external interfaces, performance, attributes (portability / correctness / maintainability / security, etc) and any design constraints imposed on the system implementation.
- **Environment of the SRS;** including scope & context.
- **Characteristics of a good SRS;** including checklist criteria of (1) correct, (2) unambiguous, (3) complete, (4) consistent, (5) ranked for importance and/or stability, (6) verifiable, (7) modifiable and (8) traceable.
- **Joint preparation of the SRS;** including participation by various members of the software development team, sales, support & training staff and clients, etc.
- **SRS evolution through refinement and future definement;** i.e. ongoing improvement
- **Product Prototyping;** establishment of the protocols, procedures and documentation for evaluation of the product through the mechanism of prototyping.
- **Embedding design in the SRS**; proposed solutions are NOT the same as requirements!
- **Embedding project requirements in the SRS;** provision of documentation cross-references and benchmarked QA criteria.

Finally, often it is extremely difficult and costly in terms of time, money and manpower to document the existing system by reverse engineering its design back into a formal Requirements Specification, however, as a viable alternative the existing 'working' version of the software may be presented as the specification including all associated user / technical documentation, QA / test scripts, source code and working install versions. Also appropriate would be printed copies of all components within the user interface; this includes screens, menus, reports and pop-up's, etc. Likewise these would also be supplemented by 'prototype' versions of any new functionality with a full write-up explaining in detail its intended operation.

Briefing on Software Usability Testing

Software Usability Testing is not User Acceptance Testing! But it is just as important …

- Within BS7925-1 and IEEE Std 610.12-1990 the term **"Acceptance Testing"** is essentially defined as the formal testing conducted to enable a user, customer, or other authorised entity to determine whether to accept a system or one of its component.

- Within ISO 9241-11 (1998) **"Usability"** is defined as the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use. With term "**Usability Testing**" being the quantified and qualified evaluation of attributes of the system relating to **"Usability"**.

In truth the testing of the Software Usability is very much subjective and open to debate; it is normally conducted as a set of consultation exercise employing a holistic approach incorporating both formal and informal techniques. This includes timed exercises, questionnaires and interviews undertaken throughout the product development from initial conception through to final completion and used to ensure the product is right from the perspective of the clients and their end users. To quote Aza Raskin, who was formerly the creative lead for the Firefox browser and head of user experience at Mozilla Labs … ***"To the user, the interface is the product"***

There is no formal definitive standard for Software Usability Testing, however, the document entitled **"ISO 9241-11 (1998) Ergonomic requirements for office work with visual display terminals (VDTs) - Part 11 Guidance on usability"** provides an excellent benchmark for implementation and indeed it is commonly cited for this task within many published articles and papers

Terms defined within ISO 9241-11 (1998) include …

- **Usability:** The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

- **Effectiveness:** Accuracy and completeness with which users achieve specified goals.

- **Efficiency:** Resources expended in relation to the accuracy and completeness with which users achieve goals.

- **Satisfaction:** Freedom from discomfort, and positive attitudes towards the use of the product.

- **Context of Use:** Users, tasks, equipment (hardware, software and materials), and the physical and social environments in which a product is used.

- **Work System:** System, consisting of users, equipment, tasks and a physical and social environment, for the purpose of achieving particular goals. The context of use consists of those components of the work system which are treated as given when specifying or measuring usability.

- **User:** Person who interacts with the product.

- **Goal:** Intended outcome.

- **Task:** Activities required to achieve a goal. These activities can be physical or cognitive. Likewise Job responsibilities can determine goals and tasks.

- **Product:** Part of the equipment (hardware, software and materials) for which usability is to be specified or evaluated.

- **Measure (noun):** Value resulting from measurement and the process used to obtain that value.

Briefing on ISO/IEC 29119 Software Testing (New International Standard)

The aim of ISO/IEC 29119 Software Testing is to provide one definitive standard for software testing that defines vocabulary, processes, documentation, techniques and a process assessment model for software testing that can be used within any software development life cycle. The standard is centred around a three-tier risk-based process model for software testing that provides guidance on the following.

- The development of organisational test strategies and policies.
- The management of testing projects including the design of project/level test strategies and plans.
- Monitoring and controlling testing.
- Dynamic test process that provides guidance for test analysis and design, test environment set-up and maintenance, test execution and reporting.

It is currently being developed, trialled and reviewed by practitioners and academics around the world, with 27 nations represented on the working group that is responsible for developing the standard.

ISO/IEC 29119 comprises 5 parts:

- Part 1: Definitions & Vocabulary.
- Part 2: Test Process.
- Part 3: Test Documentation.
- Part 4: Test Techniques.
- ISO/IEC 33063 Process Assessment Model for Software testing processes (dual standard number pending).

The standard will replace a number of existing IEEE and BSI standards for software testing:

- IEEE 829 Test Documentation.
- IEEE 1008 Unit Testing.
- BS 7925-1 Vocabulary of Terms in Software Testing.
- BS 7925-2 Software Component Testing Standard.

IEEE have given ISO permission to use IEEE 829 Test Documentation standard as a basis for part three of the new standard. ISO 29119 will eventually supersede IEEE 829. For more information on IEEE 829 refer to www.ieeexplore.ieee.org

The British Computer Society has given ISO permission to use the BS-7925-1/2 Component Testing standard as a basis for part four of the new standard. ISO 29119 will eventually supersede BS-7925-1/2. For more information on BS-7925-1/2 refer to www.testingstandards.co.uk

For more details on the ISO/IEC 29119 please see www.softwaretestingstandard.org

Briefing on ISEB Foundation Certificate in Software Testing

It is the author's recommendation that all those who are employed within Software Testing and Quality Assurance (or teach the subject) should be qualified with a minimum of at least the ISEB Foundation Certificate; for such a qualification provides an industry recognised standard that is proven to deliver the QA results sought.

The Information Systems Examinations Board or ISEB (www.bcs.org/iseb) is part of the British Computer Society which is the Chartered Professional Institution for Information Technology within the UK and is licensed by the Engineering Council UK and Science Council UK respectively.

ISEB certification leads to ISTQB certified testing qualifications. The Foundation Certificate in Software Testing now has dual accreditation with the ISTQB (International Software Testing Qualifications Board) and successful candidates will receive certification with both ISEB and ISTQB recognition at Foundation level.

The syllabus for the foundation and practitioner certificate may be downloaded from the BCS website and indeed provides excellent Teaching/Lesson Plans covering recommended tutorial content, teaching/study timescales and reference materials.

For more details of the ISEB in Software Testing see the BCS website (http://www.bcs.org/server.php?show=nav.6942).

'Quick Win' Recommendations: Short Term Implementation

- Introduce 'Component Testing', which ideally should be compiled and conducted by the software R&D team BEFORE hand-over to the QA team; even if it is only a brief précis presented on an A4 sheet detailing the nature of the functional changes and additions along with a recommended approach for testing. Likewise ideally the program code should be demonstrated by the software R&D team as tested and working without any issues being observed; where possible including the provision of test scripts and technical documentation. This provides a proven vehicle for both quality assurance and knowledge transfer with appropriate peer review. This approach would be applicable for testing of the user interface, system functionality and non-functional aspects. Essentially the resultant test scripts should permit subsequent repetition and expansion by the QA team who would then undertake the implementation from the perspective of the end-user. The test script should commence from the initial set-up and encompass the entire lifecycle of the system operation, without requiring any 'back-door' activities or system fudges. It should include initial set-up, user inputs, expected outcome and the necessary system environment along with the operational procedures to be followed in easy to follow instructions.

- Introduce a standardised QA management model which is based upon the hierarchical V-model within an incremental and iterative implementation and encompassing multiple System Configuration Permutations; constituted from 'Component Testing', 'System & Integration (S&I) Testing', 'Regression Testing' and 'Acceptance Testing'. In particular the QA focus would be upon retesting of component test scripts (which are subsequently expanded into S&I test scripts) and regression testing. The test scripts for regression testing are constituted from both component tests and S&I tests which have been combined together to form a collective script and rationalised accordingly to remove any duplications, omissions and contradictions. In addition stated methodologies can be readily employed for standardising the implementation approach and resultant communiqués.

- Introduce functional testing employing BS7925 for test methodology and IEEE829 for test structure.

- Introduce non-functional, user interface, multi-user and load performance testing, as well as, compiling suitable technical documentation.

  - Non-Functional: Pre-implementation 'Software Trials' focusing upon the operation and stability of new installations, upgrades and data conversions. Includes Multi-User and Load Performance testing from the perspective of the clients and their end-users.

  - User Interface: Ensuring correct menu and dialogue navigation with input field validation. Determining that the software application is intuitive in design, user friendly and robust in operation.

  - Multi-User: Record locking within network environments and User Access Control (UAC).

  - Load Performance: Benchmarked functionality evaluated against prior timings and database volumes.

  - Technical Documentation: Employing SSADM techniques for business analysis and systems design; including the graphical modelling of data flows, data structures and data processing functions.

'Quick Win' Recommendations: Long Term Implementation

- Introduce offshore outsourcing for repetitive testing such as regression testing (implemented as per test scripts) and also for software trials (implemented as per installation guides and system manuals). Software trials are conducted in conjunction with regression testing and are undertaken on an immediate basis without the use of test scripts; they cover new installations, updates, upgrades, data-conversions, and third party integration, etc. Through employing offshore outsourcing, the work that is time consuming and repetitive in nature can be offloaded in a cost-efficient and effective manner which permits the in-house QA team to concentrate on further developing their repertoire of technical documentation and test scripts.

- Introduce version control upon both the distributable build and source code to include the databases and any stored procedures and triggers. Misunderstandings often arise if the version being used for issue investigation does not correspond to the version in which the issue was first identified; this is compounded further if there are several divergent code branches in use. Likewise an automated custodian is recommended to administer the various versions of the source code. *An example would be Surround SCM available from Seapine Software - http://www.seapine.com/surroundscm.html*

- Introduce issue tracking systems at both the software vendor, and where possible, at the end-user sites for KPI metrics analysis. Issue tracking is extremely useful in ensuring adherence to Service Level Agreements (SLA). *An example would be Test Track Pro available from Seapine Software - http://www.seapine.com/ttpro.html*

- Introduce a set of minimum standards for technical documentation and test scripts. Ensuring that an appropriate level of detail and language is prescribed which is suitable for the intended readership; employing a format that is clear, concise and unambiguous. Later this can be expanded into a full Quality Management System (such as ISO9001:2000) that encompasses all aspects of Quality Assurance, Software Testing and Metrics production within the organisation.

- Introduce a 'quality champion' at either senior manager or company director level; who is tasked with enhancing the clients end-user experience in regards to the company, its products and services.

- Introduce regular internal audits of the QA function which would be performed in-house by a senior company official so that an appropriate assessment can be made as to the cost-effectiveness and efficiency of the department and the organisation as a whole. This would cover the staff, documentation, procedures and legal regulations employed with a view to make continual improvements. This helps to highlight any issues that need urgent addressing including none-compliance of SLA's (intentional or otherwise) and unsanctioned agendas.

- Introduce a QA website for external and internal access that provides PDF download of management approved technical documentation, training notes, test scripts, SQA metrics and release notes. These then can be used for Acceptance Testing by both QA staff and clients alike. Also such a QA website would provide a professional portrayal of the product lines functionality and stability that is open, transparent and accessible

- Introduce a 'steering group' whose responsibility is to identify and proactively address 'organisational' and 'procedural' bottle necks and obstacles. In regards to QA the flow of work from software development to testing should be managed so that is steady and appropriate; that is to say the workflow peaks, troughs and droughts should be countered so that delivered work has the opportunity to be tested and fixed in a timely and proper fashion.

'Quick Win' Recommendations: Promote Raison D'être & Justify ROI

Good software virtually sells itself to both staff and client alike, requiring only the minimum of support, training and ongoing maintenance. Indeed the implementation of software testing and its associated quality assurance is empirically proven to result in significant return on investment (ROI) that can be readily realised through short-term financial savings and long-term extended profits.

- Financial savings are realised through finding faults within software products and services before clients and competitors, thus protecting investment and reputation.

- Extended profits are realised through building brand confidence by authoritatively attesting that the products and services are as described, fit for purpose and quality assured. Thus providing the foundations upon which business success can be built and litigation mitigated.

These savings and benefits can be clearly demonstrated through the appropriate application of key performance indicator metrics (KPIs), which when coupled with the extrapolated sales statistics, demonstrate that software testing and its associated quality assurance provides a major cornerstone to business success.

Thus the ROI is achieved through financial savings over the short-term and extended profits over the long-term.

'Quick Win' Recommendations: Test Strategy in Summary (Scheme of Work)

Time Line Progression …

Tool-up & Documentation …

Testing …

- **Six monthly cycles; proving incremental and iterative phased deliveries with weekly and monthly sub-cycle progression**
- **During each cycle new and amended functionality is included within the system design; this will require continual documentation and testing.**
- **Implemented using QA Management Model; ranging in size from intrinsic through to full Quality Management Systems such as ISO9001.**

'Quick Win' Recommendations: Test Strategy in Summary (Approach)

QA Objective	Product Confidence	Product Communication
Direct Approach	Testing	Documentation
Indirect Approach	QA Procedures	Training
Intended Outcome	**Comply with Legal Benchmark …** *Sale of Goods Act 1979 as amended:* - *Conforms to stated description (inc. advertisements and contracts)* - *Of satisfactory quality* - *Fit for purposes* **Achieved by …** *Exercise & Evaluation of System:* - *Screen Dialogues & Menus* - *Reports & DB Extractions* - *Features & Functionality* - *Mathematics & Financials* - *Performance & DB Volumes* - *Operational Stability* - *Multi-User & Multi-Site* - *Scalability & Config Options*	Knowledge Transfer
Assessment	Software QA Metrics	Feedback

GOFAR Teaching Paradigm

The QA Training Materials was revised for tuition provision based upon the 'GOFAR' teaching paradigm as detailed below …

- ***GOFAR: Goal > Objectives > Framework > Assessment > Review***
  - *Employing the journey analogy throughout to provide context and scope*
  - *QA Criteria used throughout*
- ***GOAL:***
  - *Final destination we wish to achieve in our course of study*
- ***OBJECTIVE:***
  - *Interim milestones; normally linear but may be iterative so long they are incremental*
- ***FRAMEWORK:***
  - *Steps treaded along the path of learning; of which there may be several valid paths*

    Remember: Everyone has they own varying pace, abilities and temperament!
  - *Mix 'n' Match from suitable Training Materials with tuition delivered in accordance with Teaching & Lesson PLANS and stated Learning AIMS & OBJECTIVES*
  - *Action Planning using 5WH - Who, What, Where, Why, When & How*
- ***ASSESSMENT:***
  - *Initial assessments (at commencement)*
  - *Formative assessments (interim)*
  - *Summative assessments (on completion)*

    NB: Are we on track? Has sufficient progress been made? ETA to Destination?

    (ETA represents Estimated Time of Arrival)
- ***REVIEW:***
  - *Looking back on the journey, where there any lessons learned?*
  - *Can improvements be made for next time?*
  - *Feedback should be positive and none-punitive; where possible focussing upon commendations and recommendations for improvement*

The above paradigm may be adapted for a variety of sectors; including business, science, engineering and project management. It is compliant with most project management methodologies and can be used as an initial or interim stepping-stone approach.

Investigation Techniques & Communication Skills

Problem Investigation – SPIN

- Situation
- Problems
- Implications
- Needs Payoff

Test of reasonableness in terms of approach – PLAN

- Proportionate
- Legal
- Accountable
- Necessary

Communication techniques – LEAPS

- Listen
- Empathise
- Ask Questions (5WH & TED)
- Paraphrase
- Summarise & Strategise

Enclosed questions & Summary Action Planning – 5WH

- Who
- What
- Where
- Why ← Must be careful with this one as it can offend or disconcert people!
- When
- How

Open ended questions – TED

- Tell
- Explain
- Describe

Detailed Action planning - STARE

- Situation
- Task (summary)
- Actions (detail)
- Result (intended & actual outcomes)
- Evaluation

Memorandum and reporting techniques – FIR

- Facts
- Issues
- Recommendations

NB: FIR/CAD used in stepwise refinement commencing from main points of consideration.

Post evaluation review techniques – CAD

- Circumstances
- Actions
- Decisions

NB: FIR/CAD used in stepwise refinement commencing from main points of consideration.

Continued …

Prioritisation Techniques – MoSCoW

- Must
- Should
- Could
- Wont

Objectives Specification – SMART & DUMB

- SMART: Specific, Measurable, Attainable, Relevant and Time-Bound
- DUMB: Doable, Understandable, Manageable and Beneficial

Scenario Specification - PEST & SWOT

- PEST: Political, Economic, Social and Technological
- SWOT: Strengths, Weaknesses, Opportunities and Threats

Betarix Box

- **Your Attitude ➔**{affects} **➔ Your Behaviour ➔** {affects} **➔ Their Attitude ➔** {affects} **➔ Their Behaviour ➔** {affects} **➔ Your Attitude ➔** and so on.

- Remember to keep calm, take a step back and consider the impact of any actions; whether it being physical, verbal or implied.

- Focus on DEESCALATING any situations and striving towards WIN-WIN resolutions for all.

- Communication is constituted from 55% Non-Verbal Interaction, 38% Tone & 7% Spoken Word; it is not just what you say but it is also how you say it and the way you behave.

Feedback & Criticism

- MUST NOT BE INAPPROPRIATE OR UNDULY NEGATIVE! Avoid where possible Comments / Criticisms / Complaints / Condemnations. Instead focus on Commendations and Recommendations (also known as Medals and Missions); remembering to thank participants for their contribution and provide feedback that is appropriate and objective. GOFAR can be used to structure Recommendations for improvement.

- MUST NOT BE DERISORY OR PUNITIVE! Remember people's feelings and consider all the contributory circumstances.

A Thought for the Road

I would like to offer two quotes ...

- ***What monstrous absurdities and paradoxes have resisted whole batteries of serious arguments, and then crumbled swiftly into dust before the ringing death-knell of a laugh ~ Agnes Repplier***

- ***At the height of laughter, the universe is flung into a kaleidoscope of new possibilities ~ Jean Houston***

Quality Assurance and Software Testing still has far to go in terms of universal acceptance and appropriate implementation within the Computing Industry.

Regardless of the time, care and effort expended by QA professionals in their work, there will also be critics and detractors. Even though the resultant product improvement clearly speaks for itself, they will still not be silenced nor convinced. Fortunately countering such objections, rebuffs and dismissals with a little humour and a lot of patience can be an extremely effective strategy.

In my case I often portray my QA role as being a ***'Techno Entomologist'*** or ***'IT Bug Collector'***; this slightly jovial approach helps to deflect and deflate criticism and scorn. It also makes the QA role appear none threatening and less confrontational to those within Software Development, Project Management and Budgetary Control.

Likewise where possible I try to explain what I actually do in simple terms; in my case I describe my work as being essentially ***'bagging, tagging and stacking of pesky software bugs!'***; this may be rather simplistic and light hearted but it is still extremely effective in conveying the work involved in a clear and concise manner. Particularly in a format that people can readily understand and appreciate whilst overcoming any barriers or questions they may raise.

Essentially the intention is to portray QA using a fun and friendly approach to win over people's hearts and minds whilst gaining their trust and acceptance. Where necessary more technical explanations can be provided with the QA Training Materials providing an appropriate source of reference.

Biography

Eur Ing. Christopher Pugh, CEng. CSci. CITP. FBCS. FIAP.

Professional credentials within Information Systems include European Engineer registered with Fédération Européenne d'Associations Nationales, Chartered Engineer registered with the Engineering Council UK, Chartered Scientist registered with the Science Council UK and Chartered IT Professional registered with the British Computer Society. Fellow of the British Computer Society and also Fellow of the Institution of Analysts and Programmers. First started in Computing in 1986 and has since gained extensive experience across several sectors including Manufacturing, Healthcare, Finance and Construction. Standard methodologies employed include BS7925, IEEE829 and ISO9001 covering functional, non-functional, multi-user, load / performance and automated testing. Academically qualified with the BCS Professional Examinations.

Website: www.pughconsultingservices.co.uk

Fédération Européenne d'Associations Nationales
18, B-1150 Brussels, Belgium
Website: www.feani.org
European Engineer (Eur Ing)
Registration No. 30084 (2008)

Engineering Council UK
10 Maltravers Street, London, WC2R 3ER.
Website www.engc.org.uk
Chartered Engineer (CEng)
Registration No. 531625 (2005)

The Science Council UK
32 - 36 Loman Street, London, SE1 0EH.
Website: www.sciencecouncil.co.uk
Chartered Scientist (CSci)
Registration No. BCS/111/000118 (2008)

British Computer Society
1 Sanford Street, Swindon, SN1 1HJ.
Website: www.bcs.org.uk
Chartered IT Professional (CITP) & Fellow
Registration No. 09308253 (2004, 2011)

The Institution of Analysts and Programmers
Charles House, 36 Culmington Road, London, W13 9NH
Website: www.iap.org.uk
Listed on the Register of Consultants & Fellow
Registration No. 21903 (2010)

Copyright & Appropriate Usage Policy

The QA Training Materials is simply a collection of notes and standardised forms and procedures that the author has collected and developed over many years for use within academic and commercial environments.

The content is all standard QA material which the author has tried to explain as simply and clearly as possible for purposes of training and teaching the subject of Software Quality Assurance and Software Testing.

The QA Training Materials is also intended to provide a standard 'off-the-shelf' implementation template for a Quality Management System (QMS) that can be readily deployed along with a set of fully explained Case Studies, Briefings and Quick-Wins that may be used as a source of reference.

Similar QA material is openly documented and other versions are readily accessible through both established literature and on-line.

The content is pitched at all levels, ranging from students and trainees, through to experienced IT professionals and senior management; delivering subject coverage in both breadth and depth.

It is the expressed wish of the author that QA Training Materials be used and continued by the readers; hence the content of this document has been designated by the author as public-domain and royalty-free. There is no charge for download or usage. It may be reproduced without permission, however, please reference all sources appropriately and adhere to copyright laws.

.

For more details of UK copyright laws then please see the following web-link:

http://www.ipo.gov.uk/types/copy.htm