Laboratory Software Applications Development: Quality Assurance Considerations

James W. Dillard, Ph.D., Robert E. Gladd, Vicky Hutton

INTERNATIONAL TECHNOLOGY CORPORATION: OAK RIDGE LABORATORY

"... <vendor> makes no representations or warranties with respect to the contents hereof, and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose for both this manual and the product it describes. Furthermore, <vendor> reserves the right to revise this publication and to make changes in the content hereof without obligation to notify any person of such revision or changes..."

In less than a decade, microcomputers and software applications have become ubiqitous, indispensable tools in business, industry, and the sciences. The end-user faces a bewildering array of options with respect to makes, models, and peripherals, and software compilers, libraries, firmware, and applications packages. Since the foregoing standard disclaimer may be found in the product documentation of virtually every commercial microcomputer hardware and software vendor, the end-user must blend the array of options, possible algorithmic deficiencies, and system incompatibilities into a comprehensive quality product. The user assumes--usually unwittingly--the cumulative responsibility for assuring a quality output.

Given the seemingly infinite choices confronting the would-be computer user, it is imperative that acquisition of computer technology be approached with a firm foundation on quality objectives and a detailed plan of action. Commit to practicing the rules of: Plan your work. Work your plan. Document your work. This paper provides a guide to assure the quality (fitness for use) of procured computer technology as well as any internally developed software.

The first task for a prospective user of computer technologies is to carefully define the need for utilizing computers in an unique work environment. Is the computer to be utilized for process or instrument control, sample tracking, data collection, calculations, data-base management, trending, report generation, word processing, all-of-above, some-of-above, etc.? Will it be single task dedicated or multipurpose? What are the regulatory and legal requirements of the business that apply? **What is quality?** Ask the questions at all levels of the organization from top management down. If the questions are not asked prior to hardware acquisition and software application development, the answers will probably just emerge in a manner adverse to quality objectives.

To answer the question "What is quality?," one must establish the acceptance limits for Fitness For Use: the extent to which the product or service successfully serves the purpose of the user during usage. For a radiological laboratory, as an example, quality is quite tightly regulated by federal law and regulatory guides such as 10CFR50, ANSI/ASME NQA-1, USNRC Reg. Guide 4.15 and ANSI N413, to list only a few. Other considerations, such as client contractual requirements, avoidance of litigation, and commitment to professional integrity, may also be included.

Once the organizational needs and acceptance limits of quality are defined, it is time to write the quality assurance plan. If the plan is not written at this stage to direct the development of the computer application, it will surely be molded in the image of the final product at the end, in an attempt to legitimize the project. The quality assurance plan at a minimum should reiterate the project goals and quality limits and include the following major elements:

- Require that quality functions are procedurally controlled and documented;

- Establish qualification requirements for key personnel such as systems analyst, programmer, procurement officer, etc.;

- Define acceptance criteria for placing hardware/software on an approved vendor list;

- Require that the procurement process, including receipt, be documented, governed by established acceptance criteria to quality, and have an independent review and approval cycle;

- Procedurally control the development of software and/or the blending of commercial application package into user system (Utilize the guidance of ANSI N413 as an example). Include concepts of validation and verification in activities of:
  - initiation,
  - requirements definition,
  - design,
  - coding (code in quality assurance attributes such as system security, input checking, data modification tracking, output verification, maintainability and operability).
  - integration and testing (reliability),
  - installation,
  - training,
  - operation and maintenance;

- Develop a corrective actions system to identify conditions adverse to quality;

- Provide for identification and control of quality assurance records (records necessary to document the activities performed in the monitoring program should be specified in the quality assurance program).

The rest is relatively easy. Just work your plan, and document your work.

In terms of reliability, the microcomputer is a truly marvelous piece of equipment. Its basic unit of process--the instruction--is commonly referred to in MIPS, or millions of instructions per second. Normally, the billions of instructions that comprise an application execute with a faithfulness that renders the application of conventional concepts of process tolerance bounds meaningless. The low-voltage binary logic circuits in a CPU generally operate flawlessly over the life of the machine. Electro-mechanical magnetic storage media such as floppy and hard disks and drives, while more prone to data corruption owing to mechanical malfunction, also exhibit an enviable degree of reliability.

Note the adjectives "normally" and "generally." Microprocessor vendors seem to be secretive regarding their chip manufacturing QA data, and shipments of sub-standard CPUs may find their way into microcomputer assembly lines where cost pressure considerations dictate component choices. Perusing the innards of a PC evinces the far-flung international geography of the microelectronics industry: Chip sets from Malaysia, Singapore, Korea, The Philippines, Taiwan, and elsewhere abound, begging obvious questions of **documentable** quality assurance. As an example, the contamination of chips with traces of radioactive materials has been known to cause ionization phenomena which might compromise component reliability. Be cautioned, the hardware/firmware may have bugs which are vendor, model or component dependent. Product evaluations published in PC trade periodicals invariably address issues of processing speed, capacity, features, and price, but rarely, if ever, address issues of reliability. Product reliability is apparently assumed a priori by the PC trade press, license agreement disclaimers notwithstanding. Prudence dictates that the end-user thoroughly test any hardware/firmware attributes to be utilized and periodically retest to assure that conditions remain stable.

The RAM of the operating microcomputer is occupied by the operating system (PC/MS-DOS in the cases discussed here), one or more active applications programs and/or overlays, possibly one or more background RAM-Resident programs, resident device drivers, and any number of I/O data buffers. This shared RAM environment, and its potential for operational conflicts leading to data corruption, in part gives rise to the aforementioned legalistic self-inoculation vendors feel compelled to provide with their documentation, and attempts to elicit software vendor responses to queries concerning product QA verification are likely be of no avail. Letters and calls go unanswered, or a pass-the-buck posture obtains: "Well, you see, our compiler is written in Lettuce-C, (another compiler product) so you really should contact them." To pursue this line of inquiry would lead to a near-infinite regress back to the original set of punched cards or mechanical switches, given that applications are generally written in other languages which are often written in yet other languages, which themselves are borne of particular dialects of Assembly Language, which correspond to the original native binary machine language specific to the instruction sets of the various CPU chips. It is clear that, given the universal vendor disavowal of any liability for product reliability and output accuracy, the responsibilities of data processing quality assurance rest solely with the end-user. Exhaustive user testing of individual microcomputer hardware/software configurations is nothing less than an end-user QA imperative.

The most important components of laboratory data integrity, the primary focus of the following discussion, center on accuracy and precision. Reliability is a necessary but insufficient condition for assuring data integrity; after all, output can be consistently incorrect. Accuracy of output is inextricably bound up in the interplay of hardware, firmware, and software. And here, as one might expect, potential QA difficulties multiply geometrically.

Accuracy and precision issues are typically focused on the issue of numerical error owing to algorithmic deficiencies and the necessary choices that are made to truncate non-terminating fractional numbers at the floating-point significand boundaries of the microcomputer ALU. In particular, the results of error propagation through iterative processess may significantly impact reported values, and care must be taken to minimize any such effects by thorough cycles of coding and testing.

Testing begins with assessment of the numeric operators and mathematical functions [e.g., ROUND(), LOG(), SQRT(), and EXP(), and arithmetic operators] of the compiler(s) and/or interpreter(s) to be used for applications development. Since the end user will not likely have access to the product source code, verification is accomplished indirectly through the use of short iterative programs that generate tables which are then checked against authoritative standard tables, such as those contained in the CRC Handbook of Chemistry and physics. An example of a short routine for checking powers and roots follows:

```
* SQ_CU.RTS, generates squares, cubes, & roots RE:
* Handbook of Chemistry & Physics, 55th edition, pp.
* A-80,99 IT/RSL microcomputer algorithm QC check,
* DOS 2.x, 3.x, dBASE III+ (tm) Language environment
*
SET DECI TO 8
N=1
DO WHIL N<101
NSQ1=N^2
NSQ2=N*N
NSQ3=N**2
SRN1=SQRT( N)
SRN2=N^.5
SR10N=SQRT(10* N)
NCU1=N^3
NCU2=N*N*N
CRN=N^(1/3)
CR10N=(10*N)^(1/3)
CR100N=(100*N)^(1/3)
SET PRIN ON
? '<list calculated memory variables in tabular format>'
SET PRIN OFF
N=N+1
ENDDO
```

The foregoing generates a printed table that is checked against the reference document to affirm that the functions and operators are indeed working correctly. Note that many mathematical expressions, such as those yielding powers and roots, may often be expressed in more than one way, and it is advisable to check the syntactically equivalent statements for consistent output exactitude, since a programmer may use syntactical equivalencies in a random fashion during applications development, and different programmers may habitually favor one alternative equivalent expression over others. Is it in fact uniformly the case that SQRT(N) = N^.5, or that N^2 = N*N = N**2 ? A little extra coding takes it out of the potentially hazardous realm of assumption.

Similar iterative table-generating routines may easily be coded to verify trigonometric, exponential, logarithmic, statistical, financial, and numeric conversion functions, and to verify the proper functioning of the available mathematical operators of the source code language. These routines should be executed in the various hardware/operating system/software environments they will be applied to during the operation of finished applications. Do the routines yield identical results under all releases of DOS, through all CPUs (8086, 80286, 80386), with or with-floating-point numeric coprocessors present? If the source code will run in an interpreted environment, does the interpreted output result exactly match that of the compiled code? Questions such as these may only be unequivocally answered by a thorough process of coding, testing, and documenting.

It is important to note again that testing of the accuracy and precision of numerical processing, while the principal concern from most QA perspectives, is not the entire story. Data storage and retrieval consistency should be tested on a routine basis, for both numeric and string data. It should not be blindly assumed that data stored on magnetic media, be they floppy disks, hard disks, or tape drives will exhibit flawless reproducibility over time. It is a relatively simple task to develop routines which use standard input/output sets to verify storage and retrieval reliability.

Once a compiler or interpreter has been qualified in the foregoing manner for general numeric accuracy and precision, and mass storage I/O integrity, the cycle of planning/coding/testing/debugging/re-coding/retesting begins for software applications development. A thorough front-end assessment of the scope and requirements of a proposed application is an essential first step. It is no exaggeration to observe that, in a sense, the time spent de-bugging is inversely proportional to the at least the square of the time spent planning the application. Projects may often evolve into a maze of spaghetti code when prematurely written as small applications coded "on the fly" to meet an urgent processing need. This approach inevitably invites a continuing stream of requests for the program to be altered to address "just one more task" and placed back on-line A.S.A.P. Source code expanded in such a fashion will often acquire a crazy-quilt of loops, IF-THEN-ELSE statements, and boolean .AND./.OR./.NOT. modifiers that risk logic flaws difficult to trace as the source code begins to run to many pages and program logic flow becomes harder to track absent a comprehensive pre-code design.

Source-code traceable program bugs stem from either syntactical or logical errors. Code de-buggers are capable only of syntactical error flagging: Program logic is the responsibility of the design team and the programmer. Compounding the potential problems of logic de-bugging are the presence of undocumented syntactical anomalies, i.e., source code statements and functions whose executions differ from their published specifications, leading to unanticipated program flow control and processing difficulties. There are multiple major vendors of each source code language, most claiming to adhere to the ANSI Standard where one exists, but each likely to issue product containing a different particular subset of undocumented instruction anomalies. Since compilers and interpreters themselves undergo continual revision, undocumented anomalies are never centrally catalogued, most being reported by users in a sporadic, anecdotal fashion in the computer magazines. Undocumented syntactical flaws are often ones that only become apparent when a specific and relatively rare combination of program variables intersect to yield an unexpected and incorrect result, and thus may only be discovered accidentally. The programmer should note and document each finding of a source code instruction that does not perform as specified. Such documentation should include the coding alternative(s) to the problem, i.e., the work-around(s)

The programmer should, wherever practicable, employ a strategy of modular programming. There are a host of operations that are either identical, or nearly so, across applications. The programmer should develop libraries, or modules, of code that are relatively easily incorporated into new applications with little or no revision. Libraries of add-in functions and source code language extensions are widely available commercially, and may be useful in the avoidance of re-inventing the wheel during applications development (while keeping in mind the responsibility of the end-user to test purchased libraries as thoroughly as the rest of the equipment and programming tools). Where program branching or looping conditions are concerned, logic flow integrity is maintained by a coding strategy that avoids the use of potentially amphibolous boolean statements. For instance: Consider the statement **X .AND. Y .OR. Z**, a statement that could be construed as meaning (X .AND. Y) .OR. Z, *or* X .AND. (Y .OR. Z). Control statements that contain multiple boolean operators are a potential source of trouble, and should be coded and tested with great care. Where a source language allows for the use of nested IF-THEN-ELSE statements, compound boolean statements may, to a substantial degree, be avoided in favor of more a more explicit sequence of nested IF-ENDIFs.

Sufficient internal source code documentation is invaluable, especially for large applications with long development times, and for situations in which the programmer must keep a variety of projects moving along in various stages of completion every day. The key word here is **sufficient**. Examples are rife in which an observer has a difficult time picking out the actual code buried within paragraph after paragraph of verbose source code commentary. To a certain minimal extent where high level source code languages are used, the code should be at least somewhat self-documenting, with a readily apparent program flow and logically grouped blocks of structured code that give some indication of the operations being performed. Where a language interpreter or compiler ignores blank spaces surrounding commands and their arguments, it may be helpful to use source code indentation, particularly to indicate looping and branching operations. Blocks of logically related code may be separated by one or more blank lines to enhance readability.

In cases where applications perform substantial floating-point arithmetic, and particularly where mathematical operations reside in iterative loops, care should be taken to minimize, to the extent possible, the effects of rounding and/or truncation error propagation. Specifically, division and root-taking contribute significantly to floating-point error. Where coding choices are available, the number of division and root-taking operations should be minimized. Never insert rounding operations in interim calculations; Full floating-point capacity should be carried throughout a computational sequence, with the desired rounding function applied to the end floating-point result.

Where an application is designed to process user-supplied input, most often that of data entered at the keyboard, every practical check routine should be applied to the user input before any processing takes place. Two alternatives of input format are available; The item-by-item prompt, and the screen form. Most high-level applications employ user input screen forms to accept keyboard data. Typically the user brings the data to the computer on a paper form that generally mirrors the layout of the CRT form. Some high-level languages contain built-in screen template/keyboard input functions that restrict the allowable input to appropriate data types and ranges, trapping the cursor in an entry cell until the data restrictions have been met. With the item-by-item alternative, each input item may be evaluated before the next input prompt appears. Whatever the input method employed, input data should be evaluated to the maximum practical extent, with loop-back and exit options coded in to minimize the possibility of erroneous output generation. Where an application reports a processing result to a disk file or printer, it is good practice to include the original user input data set with the output set for easier verification of results.

Before concluding this paper, some examples to emphasize the importance of quality assurance considerations in computer software systems development are instructive. A personal encounter with a reputable compiler product emphasizes the importance of testing. A new revision of the compiler was purchased and tested per the QA plan. The new version egregiously failed to replicate a standard output data set from a previously compiled and qualified application. The vendor was informed in writing of the condition adverse to quality. Silence, followed by vendor restatement of the product disclaimer were the initial responses. Some time later, however, a free unsolicited corrected revision was distributed to all users on record, with no caution or explanation concerning use of the defective version.

A final example concerns a modern analytical instrument married to a microcomputer controller. The black box now contains fewer dials and switches than its analog predecessor because the proprietary software now controls the box. The user must explicity and correctly inform the controlling software of the switch position on the front panel critical to the proper functioning of the equipment: an incorrect switch setting is not flagged by the system. Equipment incompatibility and insufficient communication between critical components may result in the generation of anomalous data that is not readily apparent during process operations nor easily identified during a data review process.

In conclusion, the bottom line for quality assurance considerations rests with the end-user. Computer technology may be used as a convenience to improve productivity, but does not release the professional from the responsibilities of assuring quality. Given that each microcomputer hardware/software combination will likely be a unique one, a professional committment toward data processing quality assurance is best administered through a systematic and thorough program of documented applications development and testing. ☐

APPENDIX A:

The following is based upon a motion in an actual client case, and is typical of the type of documentation demands made by litigants in high-stakes liability actions. It should be readily apparent that microcomputer equipment and software may rightly be construed as falling within the purview of such a motion for discovery, and further underscores the prudence of a comprehensive microcomputer Quality Assurance program.

MOTION FOR DISCOVERY

"Defendant shall produce all notes, memoranda, correspondence, documents, or records of any sort (regardless of form, including magnetic media, or any other form of recording) which relate directly or indirectly to any samples or analyses from, near, or related to the property owned or occupied by any of the plaintiffs in this action. Without limitation, the documents with respect to samples collected or analysis performed shall include all drafts, copies, originals, field notes, laboratory notes, telephone or calendar notations, and computations which relate to: the manner, method, location, and number of samples to be collected, the compounds to be tested for, the analyses requested, the cost of the analyses requested, the billing for the requested analyses, and the reporting of the analytical data for the requested analyses; all analyses actually performed, whether reported or not; all documents pertaining to the methodologies or procedures used in the storage, custody, handling, preparation, extraction, analysis, and disposal of all samples whether reported or not reported, including all records pertaining to the educational and/or professional qualifications, certifications, and training of all personnel directly or indirectly involved in said laboratory analyses performed on said samples pursuant to the plaintiffs' action; the written Quality Assurance plan under which the analyses were performed; all records pertaining to specific internal quality controls, including, but not limited to replicate samples, spiked samples, split samples, blanks, internal standards, zero and span gases, quality control samples, surrogate samples, calibration stan-

dards and devices, and reagent checks; the results of all internal and external performance and system audits immediately preceding and immediately following the performance of the above analyses; the results of all EPA reference and spiked samples; all measurements of data accuracy, precision, and completeness with respect to the analyses performed; records and documents providing the manufacturer, model number, and operations manuals for all instrumentation used in the analyses; and the records of the operating conditions of all instrumentation used to perform the analyses, including the results of all maintenance and preventative maintenance performed."

REFERENCES

Title 10, Code of Federal Regulations, Part 50, Quality Assurance Criteria for Nuclear Power Plants and Fuel Reprocessing Plants

ANSI/ASME NQA-1, Quality Assurance Program Requirements for Nuclear Facilities

U.S. Nuclear Regulatory Commission, Regulatory Guide 4.15, Quality Assurance for Radiological Monitoring Programs (Normal Operations) - Effluent Streams and the Environment

ANSI/IEEE Std. 730, IEEE Standard for Software Quality Assurance Plans

ANSI/ANS 10.5, Guidelines for Considering User Needs in Computer Program Development

ANS 10.4, Guidelines for the Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry

ANSI N413, Guidelines for the Documentation of Digital Computer Programs

JAMES W. DILLARD, Ph.D. is the Technical Director and Deputy Laboratory Manager for the Oak Ridge Laboratory of International Technology Corporation.

ROBERT E. GLADD is Vice President of Corporate and Academic Media Associates, Inc. of Knoxville, TN. (CAM3). He is a microcomputer applications consultant for the Oak Ridge Laboratory of IT.

VICKIE HUTTON is the Quality Assurance Coordinator for the Oak Ridge Laboratory of IT.