

RECEIVED BY OST: JUN 16 1986

UCRL-94600
PREPRINT

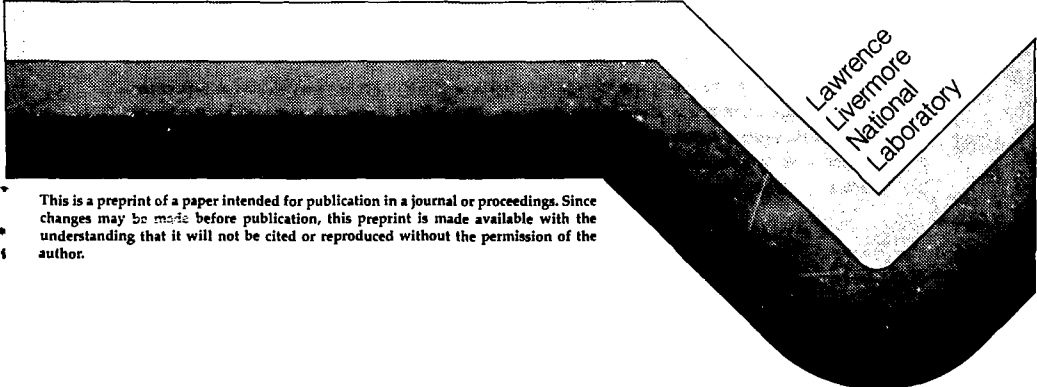
CONF-8603128--1

AN APPLICATION OF SOFTWARE QUALITY ASSURANCE
TO A SPECIFIC SCIENTIFIC CODE DEVELOPMENT TASK

John J. Dronkers

This paper was prepared for submittal to
17th California Quality Week
Santa Clara, California
March 21-22, 1986

MARCH 1986

The logo for Lawrence Livermore National Laboratory is a large, stylized 'L' shape. The top horizontal bar is white, the middle horizontal bar is dark grey with a halftone dot pattern, and the bottom horizontal bar is solid black. The right side of the 'L' is a diagonal line sloping downwards from the top right to the bottom right. The text 'Lawrence Livermore National Laboratory' is written in a sans-serif font, oriented vertically along this diagonal line. The text is white on the white background and black on the dark grey and black backgrounds.

Lawrence
Livermore
National
Laboratory

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

UCRL--94600

DE86 011640

**An Application of Software Quality Assurance
to a Specific Scientific Code Development Task.**

**John J. Dronkers
Quality Assurance Specialist**

**Nuclear Waste Management Projects
Earth Sciences Department
Lawrence Livermore National Laboratory
Livermore, CA 94550**

**Preprint - for presentation to the
17th California Quality Week
March 21-22, 1986
Santa Clara, CA.**

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Q510

1. INTRODUCTION*

This paper discusses an example of software quality assurance. The example is specific to both the software and the quality assurance measures that were developed for it. The latter are also the result of circumstances that produced the software.

This paper does not draw general conclusions. It is felt that software quality assurance has not yet developed to a point where it can lay claim to a set of rules with universal application. It is, therefore, left to the careful reader to make inferences from the material presented here and to judge the usefulness of this example to other applications.

The software considered is a scientific computer program called EQ3/6. It ". . . is a set of related computer codes and data files for use in geochemical modeling of aqueous systems." The software ". . . centers around two large computer codes, EQ3NR and EQ6, . . ." The former's ". . . function is to compute a model of the state of an aqueous solution," while the latter ". . . calculates models of changes in aqueous systems as they proceed toward a state of overall chemical equilibrium."¹

EQ3/6 is being developed by a team of scientists of the Lawrence Livermore National Laboratory's Earth Sciences Department. Its development is sponsored by several organizational entities of the Department of Energy, which will use the software for application in the nuclear waste repository program.

2. SOME ASPECTS OF EQ3/6 DEVELOPMENT

A. Definition of Quality.

The team of EQ3/6 scientists defines software quality as follows: ". . . [1] quality means that the software is useable outside the developing organization, [2] that necessary documentation to support proper usage exists, [3] that the software executes problems within the scope of its capabilities as defined by the documentation, and [4] that documentation of the validation of the models and submodels contained within the software is sufficient to allow the user to refer to it as a basis for assessing the reliability of the code for making any specific calculation."²

* Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract number W-7405-ENG-48.

For purposes of discussion, it might be useful to compare EQ3/6's definition of quality with Boehm's "Software Quality Characteristics Tree."³ Without going into the details of Boehm's grouping of attributes, it may be seen that the first clause of EQ3/6's definition corresponds to the characteristic called portability. The third clause, ". . . software executes problems within the scope . . ." closely matches the software reliability characteristic.

Clauses two and four of the definition are more difficult to compare, until one realizes that the final result of the EQ3/6 team's work is a product that consists of the software, documentation that traces the development of the software, and user manuals. In addition, the software contains internal documentation, i.e., embedded documentation. Therefore, if by documentation one assumes the entire documentation available to EQ3/6 users, then clauses two and four can be said to fit Boehm's testability, understandability, and modifiability characteristics.

The only characteristics not addressed in the definition are efficiency and human engineering. This does not mean that EQ3/6 lacks these; rather, they are not considered when determining whether or not the end product meets its quality requirements. It can be concluded that EQ3/6's definition of quality contains most of Boehm's characteristics, which, considering the specialized nature of the code, is quite remarkable.

B. Notes on Scientific Software Development.

A strong case can be made to show that scientific software is never finished. It is perhaps true that no software, scientific or other, is ever finished. However, in non-scientific software, changes in approved and productive computer programs are usually mandatory or enhancing: mandatory when bugs are discovered, or when a change in user operations necessitates software modifications; enhancing when, through application, ways are found to make the software more efficient and effective, or to take advantage of hardware or software developments.⁴

Scientific software is different. Scientific software usually starts with a desire to simulate a physical process. The physical process is translated into a mathematical model and then converted into computer algorithms. There is, however, ". . . usually more than one specific model available for any given process . . . , each with its own strengths and limitations." As the software reaches increased levels of maturity, its applications tend to increase and ". . . there is pressure to include kinds of submodels that were formerly ignored, . . ." ⁵

EQ3/6 provides a good example. It was originally developed from 1975 to 1977 "to study the interactions of sea water with basalt in mid-ocean ridge hydrothermal systems." ⁶ Since that time, several sponsors have funded enhancements and additions that have taken the code beyond its original purpose. Existing versions are in use ". . . [to study] ore forming processes, [to study] more general aspects of rock-water interaction, and in the petroleum industry." ⁷ Current code development is centered around modeling applications for the nation's first nuclear waste repository.

The continuous enhancements and addition of submodels result in software that has instead of one model, ". . . an array of linked submodels." Furthermore, the ". . . linkage [between these submodels] is often not linear, . . . [and, therefore, requires] the use of numerical (iterative) methods at the integrated code level." Testing a part of the code often requires the use of the entire code. Continuous enhancements and additions also require frequent testing to ascertain that a ". . . recent modification has not degraded the ability to solve old problems, or old kinds of problems." For that reason the EQ3/6 team maintains a verification library. "Such a library consists of a set of inputs and corresponding outputs which [sic] for a representative cross-section of the code's capabilities." ⁸

Another consequence of "endless" software development is that the development process itself is not like any of the processes described in the literature. These descriptions usually assume that ". . . [a] system development project, . . . , should be a finite, predefined, structured set of activities for the attainment of specific goals." ⁹ Once the project is completed and the ". . . program signed off, it needs to be placed under control to limit access to the production version." ¹⁰ Structured programming does not easily lend itself to a continuous software development mode; rather, a prototyping approach appears to be more efficient. ¹¹ Also, although access is controlled, it is not limited.

There are other aspects of scientific software development that make the use of traditional development models problematic. Validation, for example, is not possible in the traditional meaning of that term. Software specification requirements and software design requirements also have different implications. A software quality assurance program intended for integration with scientific software development tasks, must take careful note of the differences, lest its utility be compromised.

C. Developer Versus User.

The scientists who develop EQ3/6 are also its users. This fact negates the adage that "Those who develop and maintain computer programs are seldom the users of the programs."¹² "The situation may be further complicated by the fact that the developer, in the role of scientist, may improve existing specific submodels or propose new ones."¹³ It is as if an accountant develops a program to compute taxes, while at the same time proposing changes to the tax laws and to the principles of accounting. One serious consequence of the "developer as user" situation is that it becomes difficult to implement software quality assurance programs. The developers view the code as a ". . . calculational tool used as an intermediate step between a problem and its solution."¹⁴ This is perhaps the reason that in the past, scientific software development suffered from a lack of adequate documentation and configuration control.

EQ3/6 scientists, however, have participated in the design of the software quality assurance program to be described, and are committed to make it work.

3. DESCRIPTION OF THE SOFTWARE QUALITY ASSURANCE PROGRAM.

The software quality assurance (SQA) program integrated with EQ3/6 development is designed to demonstrate that work was accomplished in accordance with a coherent set of administrative controls. These controls are, therefore, closely tied to the creation of documents that will be part of the end product. The documents are created and collected as work progresses. They are assembled in discrete packages that allow efficient retrieval from the archives.

The SQA program consists of three major components: administrative control, configuration management and user documentation.

A. Administrative Control.

As noted in the introduction, EQ3/6 is a set of related computer codes. Its development is best described as ". . . [a team] of developers working simultaneously on several different code development activities."¹⁵ Each team of developers, or sometimes a single developer, is assigned a specific task. When all assigned tasks are completed and integrated and peer reviewed, a new version of EQ3/6 is ready to be released. The SQA program refers to the entire software development effort as "the development line" and the point of release as "the release break point". It should be noted that one development line invariably has more than one task.

At the release break point two events occur simultaneously: one, the development line is closed and, two, a new one is opened. Development lines thus provide a sequential record of development, so that if one were to call a development line "n", then its successor may be referred to as "n+1".

The reasons for opening and closing development lines are several. There may be "natural endpoints" in several tasks, which, when considered collectively, could become a natural release point for all of the software. Also, a sponsor is usually interested in, and is paying for, a specific task. When finished, the task may be ready for use, however, first it must be integrated with the rest of the software, i.e., all of EQ3/6 (as previously noted scientific software is usually a non-linear linkage of submodels). Thus, several sponsors, each with a different set of requirements, result in a development line that is most conveniently terminated when all concurrent development tasks are finished. The coordination of the development tasks, the synchronization of the task's "end points", and the decision to call for a break release point are the responsibility of the EQ3/6 Task Leader (TL).

With one development line closed, work on the next one proceeds as follows. The TL assigns a specific development task to a team of developers. For instance, there may be a task called "Add Equilibrium Sorption Model and Supporting Data Base to EQ3/6," or "Extend Precipitation Kinetics Model in EQ3/6", or "Expand and Revise Data Base."¹⁶ These tasks are the result of negotiated agreements between the TL and sponsors. In general, work on EQ3/6 falls into three categories: improvement of the current code capabilities, addition of new capabilities, or the improvement of the code's data base.

The tasks are controlled through a set of administrative procedures. Central to an understanding of the procedures is the concept of File Folder. The TL formally assigns a task by opening a File Folder. There are several types of File Folders. Consider the Code Development File Folder (CDDF). Opening a CDDF requires assigning a unique number to it, providing a title sheet that contains identifying information, and inserting an initial requirements definition and an initial personnel assignment. An opened CDDF grants the assigned team the authority to develop that specific portion of EQ3/6 described in the initial requirements definition. As each CDDF is unique to a task, so is the responsibility and authority for the task uniquely assigned.

The CDDF remains under the control of the TL, but any required documentation, apart from the initial documentation already mentioned, is submitted by the team. The type of documentation is, of course, dependent on the task. For instance, if the task involves changes to the source code, then at least three planning documents are required: one for verification, one for validation and one for documentation. If the task involves changes or additions to the data base, then only a validation plan is required. Other documentation may be "supplementary definition of the requirements, detailed planning, numerical analysis, scoping, code design, supplementary design activities, testing, . . . , letter report documentation, or revisions to previous [CDDF] documents."¹⁷

When the team completes its task, the TL closes the CDDF. All closed CDDF's are transmitted to the organization's quality assurance group for archiving. If the administrative controls were properly executed, then the CDDF contains a complete and verifiable record of the team's accomplishments. Included would be, for instance, documentation of the verification and validation activities and the references for the physical and mathematical models used. All tasks in a development line have their own CDDF and all must be closed before development line n+1 can be closed and n+2 opened.

The CDDF is an important File Folder in the development line, but there are others. There is a Maintenance File Folder. "This [file folder] authorizes changes that are restricted to the implementation of coding standards, improvements in code output, or, in the case of supporting data bases, the entry of additional or improved data."¹⁸ Other File Folders are used to authorize the application of software to an assigned problem, or the acquisition of outside codes, which may or may not be "grafted" onto

EQ3/6, or the evaluation of an acquired code. It is important to realize that the opening of a File Folder authorizes an activity; work can be done, an account can be charged.

When the work on one development line is essentially complete, all active File Folders are closed. The TL now opens four special File Folders: "Release Package Integration," 'Review Response,' 'Error Resolution' and 'Post Release Maintenance'.¹⁹

The Release Package Integration File Folder authorizes work to integrate all the tasks that occurred during the development line. Once the tasks are integrated, the Release Package Integration File Folder is closed. The next step is a peer review of the entire EQ3/6 development line. A peer review is independent of all the developers and, therefore, does not require any File Folders (File Folders are control instruments for the TL). Records of the peer review are, of course, required, but their creation, collection, and transmittal to QA is the responsibility of the Peer Review Board Chairman and is the subject of a procedure outside the scope of this paper. Any activity required as a result of the peer review, however, is again controlled by the TL. It is accomplished by the opening of a "Review Response File Folder." This authorizes work to commence on dealing with the findings of the peer review. Once the peer review findings are satisfactorily addressed, the Review Response File Folder is closed. The new version of the code is now ready to be released, or, in the language of the SQA program, the development line has reached a break release point.

The authorization to correct errors found subsequent to an official release is handled by opening, as soon as the code is released, an Error Resolution File Folder. Unlike the others, this file folder remains open as long as work continues on EQ3/6. For example, if there are five released versions of EQ3/6, then there are five Error Resolution File Folders.

One more File Folder remains, the Post Release Maintenance File Folder. "This [File Folder] authorizes changes to approved code, resulting in updates [to users], but the scope is restricted to the type of changes that are permitted under the scope of the normal Maintenance [File Folder]. A Post Release Maintenance [File Folder] is closed when the succeeding development line produces approved code."²⁰

With n+1 now completed and provided for, work on n+2 commences.

B. Configuration Management.

The SQA program's second set of controls pertains to the control of changes to approved and released versions of EQ3/6. It is called Configuration Management (CM). The development from n+1 to n+2 consists of a series of changes to n+1. None of these changes are released for application to users, the latter work with n+1 as released. However, internally the changes made to n+1 must be controlled to allow for an orderly and traceable development line. The administrative control system described in the previous section, allows the task teams a great deal of discretion and flexibility in fulfilling their assignments, which are only generally described in the requirements definition. CM allows no such discretion, nor such flexibility. Changes to approved and released versions of EQ3/6 are very rigidly controlled.

CM centers around the concept of Controlled Item (CI). A CI is any item of software, changes to which must be controlled, i.e., software items subject to CM. "Controlled items may include computer codes, supporting data files, code inputs and outputs, and documentation."²¹

The TL designates CI's. Each CI has a Code Development Log (CDL) associated with it. This is parallel to the File Folder concept, where each task has its own File Folder. Changes to the CI cannot be made unless the team obtains "development rights." Development rights are obtained by signing out the CI from its CDL. Development rights, i.e., the authority to sign out a CI, are accrued as a result of the granting of work authority provided for in the File Folder system. When the changes have been made development rights are surrendered by signing the code back in.

Controlled items have a version number. A version number has three parts. One, ". . . a release number (which identifies the development line), [two] followed by a machine letter (or decimal point), and [three] a stage number."²² A release number is assigned by the TL and remains the same for the entire development line. The machine letter identifies the specific computer to which the CI is adapted. The stage number is that portion of the version identification that changes each time a change is made to the CI. For instance, if a team signs out version 3245C17, makes changes to it, then it will sign back in version 3245C18. This translates into: a team worked on a CI 3245C,

which is a CI belonging to development line 3245, which is adapted to the Cray-1 computer; it signed out stage number 17, made changes and signed in stage number 18. The CDL reflects every single change, no matter how trivial, to the CI.

The CDL provides a sequential history of changes made to CI's. It reflects the CI's version number when signed out and the new version number when signed back in. Each CDL is identified by a name and its version number. This means that each development line creates its own set of unique CDL's. In addition, CDL's contain the name(s) of the individual who made the change and a brief description of the change.

It is possible that an assignment to a team involves more than one CI. It is therefore possible that two teams have overlapping CI's. Since CM allows only one team at a time to have development rights, there could occur a situation when one team has the development rights, while the other must wait. The practical solution to this has been that development rights are retained for very short periods of time. One might say that the teams have evolved a "swapping" method to obtain development rights.

CDL's are retained for as long as EQ3/6 remains funded and hence remains under the TL's control. Collecting the CDL's must, therefore, be initiated by the QA organization and takes the form of dated photocopies. This activity is the subject of procedures outside the scope of this paper.

C. User Documentation.

User documentation, or user manuals, takes the form of published reports. For the Lawrence Livermore National Laboratory, published reports are a principal product. Therefore, a very elaborate technical and review process has developed over the years to assure the quality and integrity of those reports. This Laboratory-wide procedure is mandatory for all projects. In addition, many projects impose additional peer reviews for their technical reports and the project of which EQ3/6 is a part is no exception.

Assignments to write user manuals are part of the work assignments made to the teams. They are reviewed by independent peers and then are subjected to the Laboratory's review procedure. Independent peer review comments and their resolution are QA records and are archived upon completion. User manuals are published by the Laboratory's Technical Information Department.

D. A Brief Comparison between the File Folder Technique and the Software Development Notebook Technique.

McKissick and Price have a described development technique called the Software Development Notebook (SDN). Briefly, the SDN is a loose leaf binder that is divided into sections. Each section corresponds to a specific phase in the software development process. Each section initially contains the requirements for its corresponding development phase and as work progresses, documents are collected in each of the appropriate sections. At the close of the development a complete records package exists. An SDN is created for each controlled item.²³

There is great similarity between the SDN and File Folder techniques. However, the development process to which each is applied is quite different. This negates some of the benefits that are said to result from the SDN technique. For instance, the File Folder technique does not necessarily result in more consistent documentation. Also, errors are not necessarily discovered at an earlier stage. The SDN technique can claim these advantages, because it makes accessible to developers the requirements of a very structured approach. The File Folder technique, on the other hand is an attempt to provide some control in a process that intentionally remains unstructured and most importantly has no ending. Once the differences are made clear, a modified SDN technique can certainly be adapted to scientific software development.

4. SUMMARY

This paper described an application of software quality assurance to a specific scientific code development program. The software quality assurance program consists of three major components: administrative control, configuration management, and user documentation. The program attempts to be consistent with existing local traditions of scientific code development while at the same time providing a controlled process of development.

5. FOOTNOTES

1. Thomas J. Wolery et al., "EQ3/6: Status and Applications", UCRL-91884, (Livermore, CA: Lawrence Livermore National Laboratory, 1984) pp. 1 and 2.
2. Thomas J. Wolery, "Requirements for Development and Use of Scientific and Engineering Software," 14 February 1986, Draft Procedure 033-SRP-P19.0, Lawrence Livermore National Laboratory, Livermore, CA.
3. Characteristics of Software Quality, cited by Joseph L. Podolsky, "The Quest for Quality," Datamation, March 1, 1985, pp. 119-125.
4. William C. Mair, Donald R. Wood, and Keagle W. Davis, Computer Control and Audit, (Wellesley, MA: Q.E.D. Information Sciences, Inc., 1978) pp 253-254.
5. Wolery, "Requirements," p. 5.
6. Dana Isherwood and Thomas J. Wolery, "EQ3/6 Geochemical Modeling Task Plan for Nevada Nuclear Waste Storage Investigations (NNWSI)", UCID-20069, (Livermore, CA: Lawrence Livermore National Laboratory, 1984) p. 4.
7. Wolery, "Requirements," p. 2.
8. Ibid. pp. 6 and 11.
9. Mair, Wood, and Davis, Computer, p. 256.
10. J. A. Burgess, "Quality Assurance for Computer Software," Quality, September, 1979, p. 111.
11. Wolery, "Requirements," p. 6.
12. Robert Dunn and Richard Ullman, Quality Assurance for Computer Software (New York: McGraw-Hill Book Company, 1982) p. 213
13. Wolery, "Requirements," p. 6.
14. Lynn C. Lewis, John J. Dronkers, and Bo Pitsker, "Control of Research Oriented Software Development", UCRL-94031 (Livermore, CA: Lawrence Livermore National Laboratory, 1985) p. 3.
15. Thomas J. Wolery, "Development of Computer Codes", 13 November 1985, Draft Procedure 033-SRP-P19.3, p. 4, Lawrence Livermore National Laboratory, Livermore, CA.
16. Isherwood and Wolery, UCID-20069, pp. 11, 13, and 14.
17. Wolery, "Development," p. 4.

18. Ibid., p. 5.
19. Ibid.
20. Ibid., p. 7.
21. Wolery, "Requirements," Appendix 1, November 1985, p. 3.
22. Wolery, "Development," p. 5.
23. John McKissick and Robert A. Price, "The Software Development Notebook, A Proven Technique," Quality, January 1980, pp. 18-22.