LBL-34423
UC-800

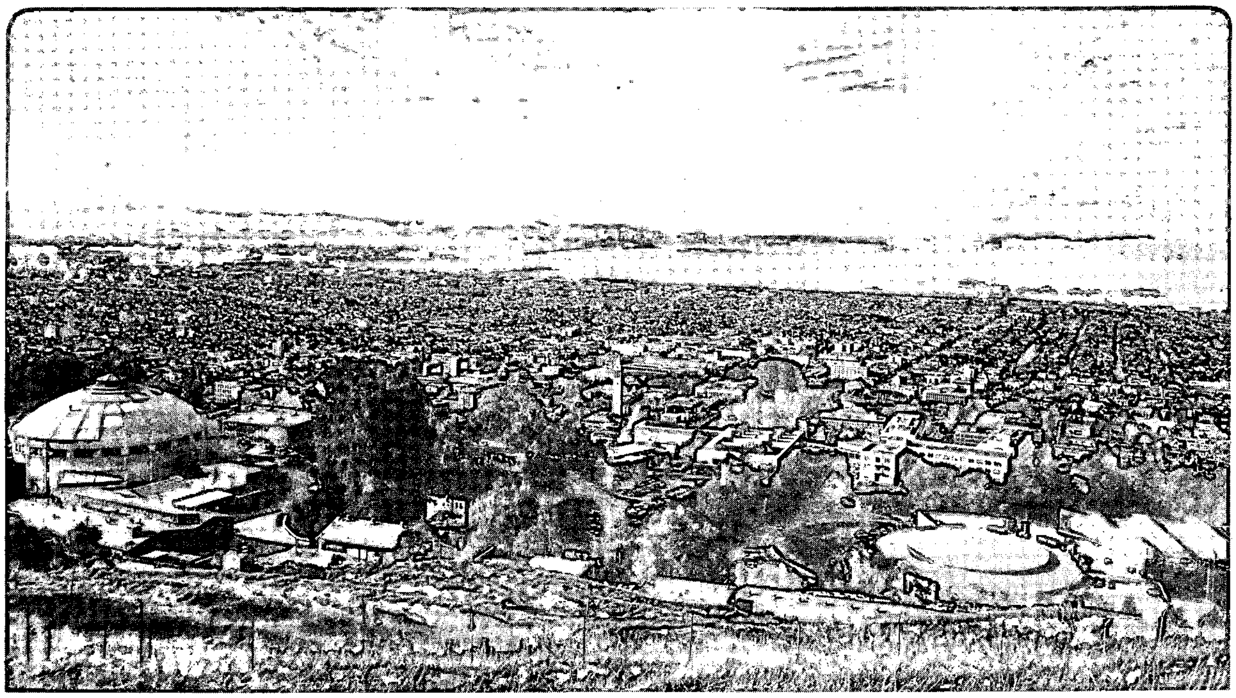
Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

EARTH SCIENCES DIVISION

Qualifying Codes under Software Quality Assurance: Two Examples As Guidelines for Codes That Are Existing or under Development

D. Mangold

May 1993



Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098

LBL-34423

Qualifying Codes under Software Quality Assurance: Two Examples As Guidelines for Codes That Are Existing or under Development

Don Mangold

Earth Sciences Division
Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

May 1993

This work was supported by the Director, Office of Civilian Radioactive Waste Management, Office of Facilities Siting and Development, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

Summary

Software quality assurance is an area of concern for DOE, EPA, and other agencies due to the poor quality of software and its documentation they have received in the past. This report briefly summarizes the software development concepts and terminology increasingly employed by these agencies and provides a workable approach to scientific programming under the new requirements. Following this is a practical description of how to qualify a simulation code, based on a software QA plan that has been reviewed and officially accepted by DOE/OCRWM. Two codes have recently been baselined and qualified, so that they can be officially used for QA Level 1 work under the DOE/OCRWM QA requirements. One of them was baselined and qualified within one week. The first of the codes was the multi-phase multi-component flow code TOUGH version 1, an already existing code, and the other was a geochemistry transport code STATEQ that was under development. The way to accomplish qualification for both types of codes is summarized in an easy-to-follow step-by step fashion to illustrate how to baseline and qualify such codes through a relatively painless procedure.

Acknowledgment

The author gratefully acknowledges the assistance of Sam Horton and John Matras of YMPO QA Division in discussions of QA as applied to software, to software developers Karsten Pruess and Chalon Carnahan for their unstinted cooperation in qualifying their codes, and to Chin-Fu Tsang for the encouragement to produce this report. This work was sponsored by the Director, Office of Civilian Radioactive Waste Management, Office of Facilities Siting and Development, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

Qualifying Codes Under Software Quality Assurance: Two Examples as Guidelines for Codes That Are Existing or Under Development

Introduction: *The Reasons for Software QA*

Software quality assurance is an area of growing concern for DOE, NRC, EPA and other agencies in recent years, due to the poor quality of software supported by them and the associated documentation on development and use. A Government Accounting Office report (GAO, 1980) reviewing the situation stated that 45% of contracted software could not be used, 19% had to be reworked, and only 2% was usable as delivered. NRC reports (NRC, 1987) showed cases where no one knew which version of a code had been used in a reactor safety analysis calculation, and others where unverified codes were used. Such sloppy scientific practice is no longer acceptable.

Actually, good software QA is just proper scientific practice. We have quality but we need to give others assurance by having sufficient records to satisfy the community at large and our funding agencies that we are following good practice.

Understanding Software Development Concepts and Terminology

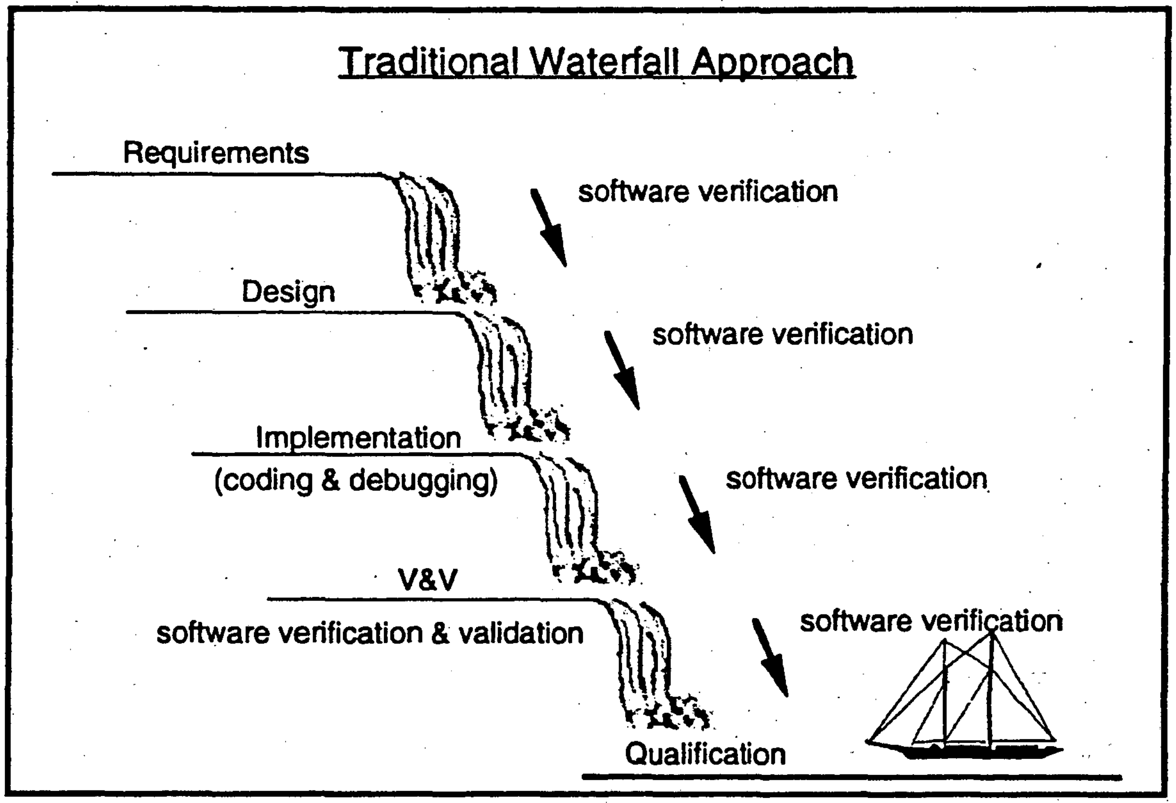
In order to understand software QA ideas and terms, there is a need to know the computer science and software industry approach to code development. This is because the software industry has been developing the principles of software QA for some years, and their concepts and experience were taken by DOE and other agencies as the basis for scientific software QA requirements.

Software development is considered by software professionals (including computer science professional societies such as the Institute of Electrical and Electronics Engineers, IEEE) to have a "life cycle" with well-defined stages (see Figure 1). Basically, the stages begin with setting up the requirements the code is to meet, then establishing the design based on the requirements, followed by the implementation of the design including actual coding and debugging, and finally the testing of the code, called verification and validation or "V & V" (see table below). This validation is not the model validation in the literature of hydrology (see below). After the code has been documented to have successfully passed through all these stages, it is considered qualified for use. In the context of software QA, this implies official recognition that the code has been properly checked both as to the meeting of the original requirements as well as the actual operation of the code itself.

<u>Summary of the Stages of the Life Cycle of Software Development</u>

Requirements	Design	Implementation	V & V	Qualification
Specifies what code must do	Structure: control and data flow	Coding and debugging	Testing and checking the code	Official QA acceptance of code for work

Figure 1. Software Life Cycle —Traditional Approach



In the professional software view, the satisfying of the original requirements is as important or more important than making sure the code runs well. They have good reason for this attitude because careful cost studies have shown that the cost of correcting an error in software goes up by as much as two orders of magnitude from the beginning of the life cycle to the end (see chart below, after Bryant and Wilburn, 1987). In other words, if an error is caught in the requirements or preliminary design stage, the cost to correct it is approximately 1% of what the same error costs if caught only after the code is in operation. This is why professional software organizations say that a software QA program pays for itself many times over by catching errors early, and its cost is only about 3-5% of the total development costs (Boehm, 1981; Wilburn, 1992).

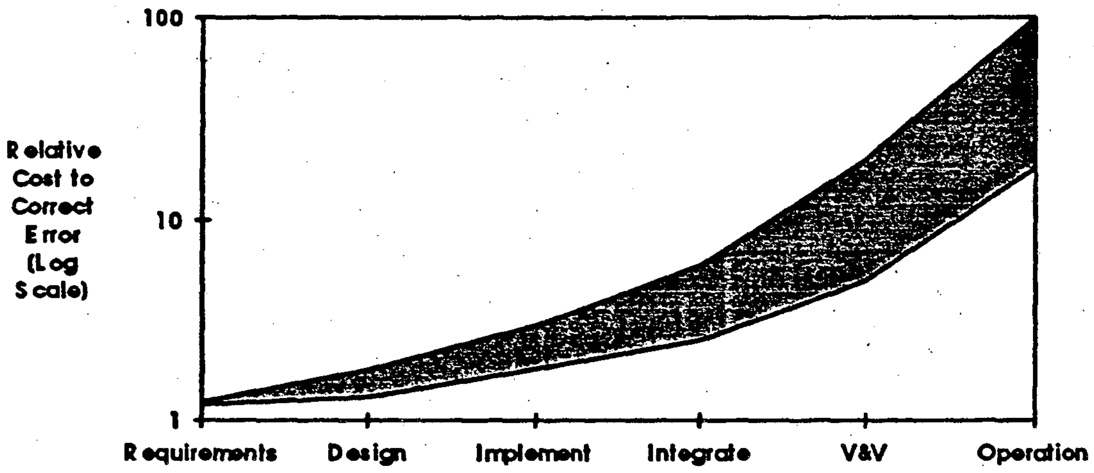


Figure 2. Relative cost to correct software errors according to stage of software development (after Bryant and Wilburn, 1987).

However, many scientists felt that this approach was too restrictive and more suited for an engineering or software industry large-scale production environment rather than a research environment where there is typically only one or a handful of code developers for a given code and goals are not rigidly set beforehand but are reached through some amount of trial and error experimentation. In DOE/OCRWM this fortunately led to the formation of the Software Advisory Group (SAG) with members from the participant organizations including LBL. Through SAG and through the desire of LBL's Earth Science Division Geologic Repository Project (GRP) to formulate a software QA plan acceptable both to scientists and to DOE, the requirements have recently been made much more flexible.

Some of the terms in software development also need explanation because they are different from the common scientific usage. Especially the terms "software verification" and "software validation" are apt to be misunderstood. Software verification means checking to ensure that a following stage (e.g., implementation) fulfills the requirements of a preceding stage (e.g., design). It does not merely mean that the mathematical formulas embodied in the code are correctly computed. Also, software validation means that the operational code satisfies all the requirements specified for it in the requirements stage. It does not mean that the code is able to match experimental or field data—that is called "model validation." For a fuller discussion of model validation, see Tsang (1987, 1989).

An Approach to Scientific Software QA

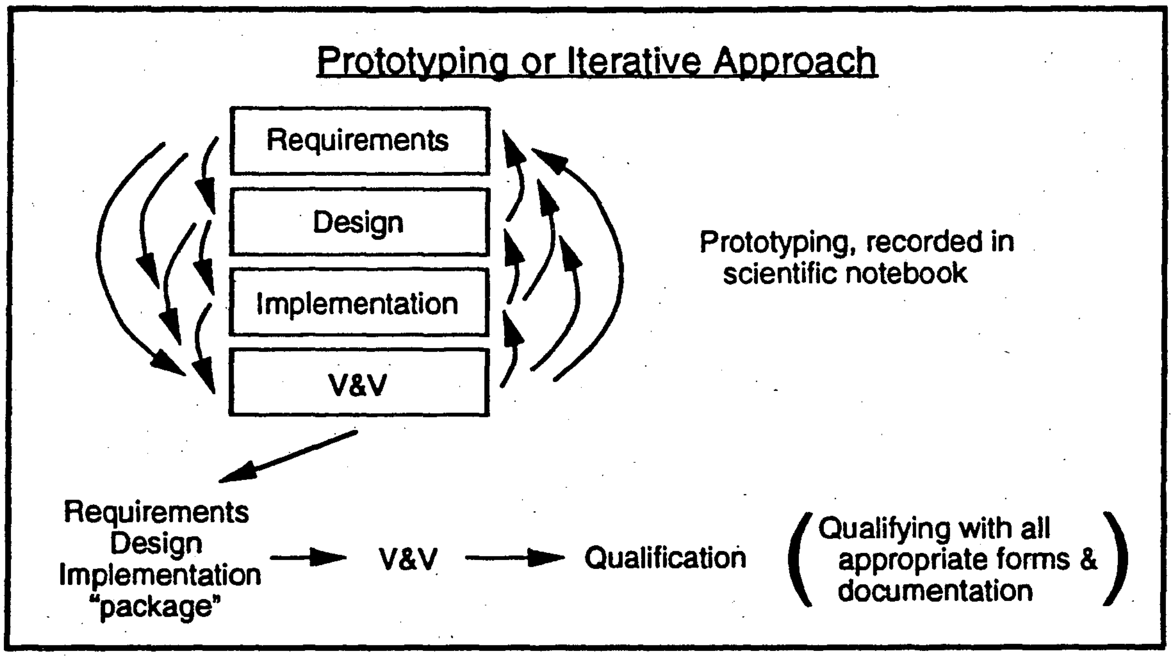
The professional software emphasis on requirements first has implications that may not fit scientific software development. The traditional software engineering approach to code development employs the "waterfall" model (see Figure 1): requirements are first established in a detailed and thorough way, then the design similarly, and so forth down through the life cycle stages like a waterfall cascading down from one level to another. One hopes that at the end there is smooth sailing.

The difficulty with this view is that the waterfall never flows back up. But scientific code development has an iterative "feedback" nature where later work may cause the developer to go back to restate the requirements or design for the code. This is called "prototyping" a code by software professionals, and within LBL's GRP project it is called the "prototype activity," where the developer can go through all the life cycle stages more than once before settling on what the requirements, design, etc., should be (see Figure 3). It is a useful way of programming when the problem to be solved is not fully defined at the outset and is the approach that fits programming for scientific research more closely. This been implemented in the GRP Software QA Plan that has been officially accepted by DOE/OCRWM (GRP, 1992) and is also incorporated into the recent DOE/OCRWM Quality Assurance Requirements Document (DOE, 1992), the latter due to the efforts of SAG.

Reasonable software QA should be based on our own existing ***proper*** practice, not just arbitrary rules. The GRP Software QA Plan and implementing procedures were written locally according to DOE/OCRWM (DOE, 1992) and LBL lab-wide guidelines (LBL OAP, 1993) for what needed to be addressed. We learned from both the DOE and LBL QA staff that the procedures we write should be based on what we actually do, not what we thought they wanted, and not to put in excessive detail nor to over-require ourselves. You will be audited to what you require yourself to do.

Under a proper QA plan, we can clean up our sloppy work habits so that we can demonstrate to others the quality of our work. A proper scientific practice keeps sufficient records for ***traceability*** and ***reproducibility*** of our results. It is no longer adequate to excuse ourselves by saying that we know what we did (even ten years ago?) or that we have been doing it this way for years (but where are those outputs from way back then?) or that "It's all in the paper" (but what version of the code did I use that time—I never made a note). We need records of essential input files, mesh designs, computer runs, etc., for our own sake to know what we have done, and to be able to properly defend our work in front of the public who may want to use our results. A ***well-kept scientific notebook*** meets most of the need for day-to-day record-keeping, and should be a normal part of all scientific work in any case.

Figure 3. Software Life Cycle —Prototype or Iterative Approach


Prototyping or Iterative Approach
Requirements
Design
Implementation
V&V
Prototyping, recorded in scientific notebook
Requirements Design Implementation "package"
V&V
Qualification
(Qualifying with all appropriate forms & documentation)

What follows is taken from the GRP Software QA Plan and implementing procedures (GRP, 1992) for qualifying a scientific simulation code, the most important and common case for our work. It has been tailored to fit our scientific programming practice and nevertheless satisfy DOE requirements. The requested documentation and other information are quite typical of what is required by other parts of DOE or other federal agencies for software QA.

Qualification of an Existing Code: TOUGH

Qualifying codes under software quality assurance is not difficult once the basic approach is understood. For illustration, a large simulation code, TOUGH (Pruess, 1987), was chosen. It already has gone through many tests (Moridis and Pruess, 1992) and has considerable documentation. Based on these it took only one week to go through the procedure to have the code baselined and qualified. The short summary below is intended as a guide to the step-by-step procedures that can serve as examples to follow in qualifying "matured" codes.

An existing code such as TOUGH has, in essence, already passed through all the stages of the software life cycle: defining the requirements, establishing the design, coding and debugging, and testing of the code, even though these terms were not used in the process. Thus, qualification is a matter of documenting what has been accomplished in a manner fitting the QA requirements. The GRP Software QA Plan has a streamlined procedure for such documenting of an existing code by summarizing what has been done on a series of forms, each of which covers some part of the software life cycle. Typical forms for qualifying an existing code are listed below with step-by-step guidance to complete them, and following this the forms for TOUGH are reproduced as an example in Appendix A.

1. ***Software Classification Form***

This is the entry form for all software, existing, acquired, or under development. For this form, as well as all those that follow, someone in the software management of the project or the division signs the form to indicate review and approval at each stage of the life cycle of the code.

- Fill in your name, date, the original name of the code and its version.
- The purpose should be just one or two sentences, but it should convey the major problems the code was developed to solve and its main applications.
- Under type, check the box that is appropriate for the code:
  - Simulation codes: complex codes for modeling physical, chemical or geological phenomena and systems, often with numerical methods.
  - Data analysis software: codes that algebraically manipulate data to make it more usable or understandable, whose functions cannot be exactly verified in all calculations performed on the data.

— Auxiliary software: codes that perform specific pre- or post-processing tasks such as interpolation, simple statistical manipulations, etc., whose functions can be verified by examining the code or hand calculation.

— System software: software that has no life cycle controls under the project software QA plan.

2. *Evaluation Report*

In Part 1, fill in the spaces as noted below.

- Fill in your name, date, the original name of the code and its version. The source can be "UCLBL-ESD."
- The intended use should be just one or two sentences describing the studies the code was developed to perform and its main application(s).
- The performance specifications should be the brief and definite requirements for performance that have been subjected to testing already.
- The adequacy of documentation is evaluated mainly by whether a complete user's manual exists, including a description of the mathematical models and numerical methods, and a code listing.
- The adequacy of software validation is evaluated by sample problems or calculations that are sufficient to demonstrate that the code meets all the performance specifications. TOUGH had 6 sample problems in the user's manual and further verification and validation problems in another report to show the code's capabilities to meet its requirements. References can and should be made to the reports which incorporate the documentation and sample calculations.

In Part 2, fill in the top as above.

- For installation and checkout, it is probably simplest to write a report summarizing the testing of the code and make it an attachment to the form, referred to on the form. One easy format to follow is shown in the report for TOUGH: list each performance specification and then give the test results that relate to it, in order, giving references to reports where the actual results are recorded.
- Under test conditions and results, make a brief conclusion of the results and how the performance specifications have been shown to be satisfied.

3. *Use Qualification Report*

This is the last form required for qualification. It provides a summary of the limitations on the use of the code.

- Fill in your name, date, and the name of the code. The baseline ID for the initial qualification of the code is just "1.0.a."

- Write any general limitations and conditions on code use. For TOUGH, this was the fact that the MA28 solver was proprietary and could not be used separately.
- Put in the limits on parameter values; a simple range is sufficient (e.g., temperature from 2°C to 360°C).
- Record limits on boundary and initial conditions. In TOUGH there was the limitation that time-dependent Dirichlet boundary conditions could not be input directly, but could be input indirectly; a reference to a report was given.

The example forms for TOUGH show a basic format to follow, and are attached for reference following in Appendix A.

After qualification, certain arrangements are made by the software management system of a project or of the Earth Sciences Division to preserve a record of the code, its documentation, and the software QA documents. The code would normally be written to a read-only file on one of the local computer systems under its baseline ID and not changed further unless it goes through formal change control procedures. The software management system also keeps copies of the user's guide and other documentation as well as the Software QA forms in its files. These records are also updated through formal change control procedures to ensure that all subsequent changes to the baselined and qualified code are carefully documented as the code is modified, errors are discovered and corrected, and so forth. Then whenever a user employs the code, they can know exactly what version they are using and what is its present status.

Qualification of a Code Under Development: STATEQ

Qualifying a code under development in software quality assurance is also straightforward once the basic procedure is clear. For illustration, a medium size geochemistry simulation code, STATEQ (Carnahan, 1993), was selected. Although it had not gone through many tests and did not have a long history of documentation, it was nevertheless not difficult to have the code baselined and qualified. The short summary below is intended as a guide to the step-by-step procedures that can serve as examples to follow in qualifying a code that is under development.

The GRP Software QA Plan employs a streamlined procedure for the documenting of a code under development by summarizing what has been accomplished on a series of forms, each of which covers some part of the software life cycle. It does not necessarily mean that each form is filled in when the code is just beginning to be developed. As explained above, scientific programming generally follows an iterative life cycle rather than a "once for all" or waterfall type of life cycle. A code under development such as STATEQ may pass through the stages of the software life cycle a few times (defining the requirements, establishing the design, coding and debugging, and testing of the code) before any stage, including the first, is ready for QA documentation (see Figure 3).

Only after the whole process is reasonably complete, and the developers have satisfied themselves that the code meets an acceptable scientific standard, do they need to begin the software QA procedures for qualification. Thus, even a code under development may be baselined and qualified with a modest effort, largely drawing on the work of development that has already been accomplished, and simply documenting it in a manner fitting the QA requirements. Typical forms for qualifying a code under development are listed below with step-by-step guidance to complete them, and following this the forms for STATEQ are reproduced as an example in Appendix B.

1. ***Software Classification Form***

This is the entry form for all software, existing, acquired, or under development. For this form, as well as all those that follow, someone in the software management of the project or the division signs the form to indicate review and approval at each stage of the life cycle of the code.

- Fill in your name, date, the original name of the code and its version.
- The purpose should be just one or two sentences, but it should convey the major problems the code was developed to solve and its main applications.
- Under type, check the box that is appropriate for the code:
  - — Simulation codes: complex codes for modeling physical, chemical or geological phenomena and systems, often with numerical methods.
  - — Data analysis software: codes that algebraically manipulate data to make it more usable or understandable, whose functions cannot be exactly verified in all calculations performed on the data.
  - — Auxiliary software: codes that perform specific pre- or post-processing tasks such as interpolation, simple statistical manipulations, etc., whose functions can be verified by examining the code or hand calculation.
  - — System software: software that has no life cycle controls under the project software QA plan.

2. ***Software Requirements Specification (SRS)***

This is the first form required for qualification of codes under development. It provides a summary of the limitations on the use of the code.

- Fill in your name, date, the original name of the code and its baseline ID. The baseline ID for the initial qualification of the code is just "1.0.a."
- For listing the requirements, it is probably simplest to write a separate document and make it an attachment to the form, referred to on the form. One easy format to follow is shown in the example form for STATEQ: list each requirement specification under the three major headings of functional requirements, performance requirements, and interface requirements.
  - — Functional requirements: these are the functions the software is to perform. In STATEQ, these included calculating equilibrium concentrations from a given set of basis species, how activity coefficients are to be estimated, two forms for treating redox equations, etc.

— Performance requirements: these are the attributes of the software such as the format and language (e.g., FORTRAN 77), issues such as portability, correctness, maintainability, etc., and the applicable stages of the life cycle.

— Interface requirements: referring to the code's relationship with other software and the operating system. For STATEQ, this meant that it was to be capable of stand-alone operation and able to write a general data file that could be utilized by another program.

One tip for making it easier for yourself later on in the process is to write down the tests of the requirements as you write the requirements. This helps you to both clarify and make practical the requirements you actually want, and it becomes a preliminary version of the software validation test plan (see below).

3. *Software Design Document (SDD)*

This form gives the major components of the design related to the requirements for the code as specified in the SRS. The document is again best written up as a separate report to be attached to the SDD form. A flow chart should be attached as part of the report.

The general content of the STATEQ SDD is very practical.

- Use the major headings of the SRS to provide the overall framework: major components of the design related to the functional requirements, the performance requirements, and the interface requirements.
- Elaborate on each item under a major heading, specifying the design of the code to meet that requirement. For example, tell how the mass balance will be met, and in the case of STATEQ, what options exist for the code's calculations of mass balance in different ways.
- The SDD should also contain a description of the physical and chemical phenomena being modeled, the equations and notation used, assumptions, simplifications, and solution techniques. The simplest way is to refer to the user's guide for all these matters.
- The SDD is also required to state the ranges of inputs and outputs. A separate small section or table may do this easily.

The sample report for STATEQ has a good format to follow for the major components of the design related to the functional requirements. It begins with the program function and flow, goes on to the mathematical problem to be solved, then to the procedure to solve the equations, and finally covers any auxiliary calculations (including optional calculations).

4. *Software Procedure Verification Summary*

This form provides a summary of the work on the code up through implementation, including requirements, design, coding and debugging. It verifies that the requirements set forth in the SRS have been carried through from one life cycle stage to the next, and that this has been documented. This is done by someone in the software management system, not the developers of the code.

- The developers just fill in the name of their code, its baseline ID, and their names at the top of the form.

5. *Software Validation Test Plan (SVTP)*

As stated above, software validation is a software development term for assuring that the requirements set forth in the SRS are met by the completed code. It does not mean that the code is compared to experimental or field data—that is called "model validation," and is dealt with later, after the code has been baselined and qualified.

The software validation test plan is prepared by the developers based on the requirements in the SRS and the design in the SDD. Again, it is simplest to write a separate report and attach it to the form. The test cases should be formulated for the inputs and boundary conditions necessary to exercise the code, and may be the same as sample problems in the user's guide.

- Follow the main headings of the SRS in order: functional, performance, and interface.
- Say what kind of test or inspection needs to be done to assure that each requirement under each of the headings has been met. The general means is testing of the code with inputs that exercise the specified functions. Some aspects such as language, modularity, use of comment lines, etc., can be checked merely by inspection of the code.

See the attached SVTP for STATEQ for a number of examples of how all these points are easily done.

6. *Software Validation Test Results (SVTR)*

The software validation testing should be performed by a person not involved in the development of the code. A summary report of the test runs and inspections should be attached to the SVTR.

- This is one form that the developers do not need to fill out.

7. *Use Qualification Report*

This is the last form required for qualification. It provides a summary of the limitations on the use of the code.

- Fill in your name, date, and the name and baseline ID of the code.
- Write any general limitations and conditions on code use. For STATEQ, these were the facts that no transport calculations are performed, nor are any kinetic calculations done.
- Put in the limits on parameter values; a simple range is sufficient (e.g., for STATEQ, the temperature can vary from 15°C to 100°C).

- Record limits on boundary and initial conditions. In STATEQ there was the limitation that no solutions of high ionic strength (brines) can be calculated ($I < 0.1$ M).

The example forms for STATEQ show a basic format to follow, and are attached for reference following in Appendix B.

After qualification, certain arrangements are made by the software management system of a project or of the Earth Sciences Division to preserve a record of the code, its documentation, and the software QA documents. The code would normally be written to a read-only file on one of the local computer systems under its baseline ID and not changed further unless it goes through formal change control procedures. The software management system also keeps copies of the user's guide and other documentation as well as the Software QA forms in its files. These records are also updated through formal change control procedures to ensure that all subsequent changes to the baselined and qualified code are carefully documented as the code is modified, errors are discovered and corrected, and so forth. Then whenever a user employs the code, they can know exactly what version they are using and what is its present status.

References

Boehm, B. W., 1981. Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ, 767 p.

Bryant, J. L., and Wilburn, N. P., 1992. Handbook of Software Quality Assurance Techniques Applicable to the Nuclear Industry, NUREG/CR-4640.

Carnahan, C. L., 1993. Computer Program STATEQ: A User's Manual, LBL-34007.

Department of Energy (DOE), 1992. Quality Assurance Requirements and Description (QARD) Document, Revision 0, December 18, 1992, DOE/RW/0333T.

Geologic Repository Project (GRP), 1992. Implementing Procedures for Software Quality Identifiers (IP 19.01), Software Control Master Log (IP 19.02), and Technical Calculations (IP 19.03), all Revision 0, November 11, 1992.

Geologic Repository Project (GRP), 1992. Quality Assurance Program, Revision 0, April 7, 1992.

Geologic Repository Project (GRP), 1992. Software Quality Assurance Program, Revision 0, April 7, 1992.

LBL Operating and Assurance Program (OAP) Plan (PUB-3111), Revision 3, 1993. Program Element 3, Work Processes—Section 3.11, Computer Software Control.

Moridis, G. J., and Pruess, K., 1992. TOUGH Simulations of Updegraff's Set of Fluid and Heat Flow Problems, LBL-32611.

Nuclear Regulatory Commission (NRC), 1984–1987. Licensee Contractor and Vendor Inspection Status Report–Quarterly, January 1984 through September 1987, NUREG/CR-0040, Vol. 8–11.

Pruess, K., 1987. TOUGH User's Guide, LBL-20700.

Tsang, C. F., 1987. Comments on Model Validation, Transport in Porous Media, Vol. 2, No. 6, pp. 623-630.

Tsang, C. F., 1989. Tracer Travel Time and Model Validation, Radioactive Waste Management and the Nuclear Fuel Cycle, Vol. 13, No. 1–4, pp. 311-323.

Wilburn, N. P., 1992. Software Quality Assurance for the Nuclear Industry, Washington State University at Tri-Cities, September 15–17, 1992, course notes.

Appendix A:

Sample Forms for TOUGH

Software Classification Form

Report prepared by: Karsten Pruess Date: March 2, 1993

Name of code: TOUGH Version: 1987

Purpose: Multi-dimensional simulation of coupled transport of water, vapor, air, and heat in porous and fractured media applicable to both site characterization and performance assessment for nuclear waste isolation, as well as geothermal reservoir studies and unsaturated zone hydrology.

Type

- [✓] Simulation code
- ☐ Data analysis software
- ☐ Auxiliary software
- ☐ Systems software

Origin

- ☐ New development software
- ☐ Acquired software
  Source:
- [✓] Existing software

Review completed and approved:

Donald Mangold 3/4/93
(Review Manager) (Date)

Evaluation Report — Part 1

Report prepared by: Karsten Pruess Date: March 2, 1993

Name of code: TOUGH Version: 1987

Source: UCLBL-ESD

Intended use: Simulation studies for site characterization and performance assessment in nuclear waste isolation.

Performance specifications: For studies in partially saturated fractured rock, the code must be capable of the following: 1) at least 2-D calculations; 2) multiphase flow of water, vapor, and air; 3) nonisothermal heat flow to temperatures above 100°C; 4) coupled transport of mass (two phases) and heat; 5) modeling of both porous and fractured media.

Adequacy of Documentation A complete user's guide is available, including a description of the mathematical models and numerical methods, and a set of 6 sample problems (Pruess, 1987).

Adequacy of Software Validation The user's guide contains 6 sample problems and a recent report contains 5 verification problems and 3 validation problems (Moridis and Pruess, 1992). Also, the latter gives references to other verifications in the published literature.

Review completed and approved:

[illegible] 3/4/93

(Review Manager) (Date)

LBL-20700
A-4

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

EARTH SCIENCES DIVISION

TOUGH USER'S GUIDE

K. Pruess

June 1987

Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098

LBL-32611

TOUGH Simulations of Updegraff's Set of Fluid and Heat Flow Problems

by

George J. Moridis and Karsten Pruess

Earth Sciences Division
Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

November 1992

Evaluation Report — Part 2

Report prepared by: Karsten Pruess Date: March 2, 1993

Name of code: TOUGH Version: 1987

Installation and Checkout: NA (Existing code developed at LBL.)

Test conditions and results. Attach or refer to documents as needed.
See attached report. According to this report, testing has been adequate to exercise the code in all the requirements given in the performance specifications in Part 1, and the code has performed satisfactorily in all tests. Also, the user's guide and the additional report are adequate to fulfill all the requirements of this Software QA Plan when the code listing is included in the documentation as it has been.

Therefore, it is determined that no further testing or documentation is necessary to fulfill the requirements for this code and it may be qualified for use.

Review completed and approved:

Donald Mangold (Review Manager)

3/4/93 (Date)

Evaluation Report for TOUGH

Test Conditions and Results

According to the performance specifications in Part 1, the code must be capable of: 1) at least two-dimensional calculation; 2) multiphase simulation including water, water vapor, and air; 3) nonisothermal computations to temperatures greater than 100°C; 4) coupled transport of mass (two phases) and heat; 5) simulation of the above processes in porous and fractured media. Each point has been tested as described below.

1. TOUGH performs its computations by the integral finite difference method which is capable of simulations in one, two, or three dimensions (Pruess, 1987, pp. 7, 9). Sample problems 3 and 5 were calculated in two dimensions in Pruess (1987).
2. The code has been designed and written to simulate multiphase flow of water, water vapor, and air (Pruess, 1987, pp. 2, 4-6). Sample problems 1, 5, and 6 in Pruess (1987) demonstrate the capability of TOUGH to model multiphase flow of water, vapor, and air including phase transitions between liquid and vapor phases. Sample problems 2 and 3 in Pruess (1987) show the ability of the code to simulate two-phase flow of water (liquid and vapor).
3. This code has been developed to calculate heat flow by conduction and convection with temperatures greater than 100°C (Pruess, 1987, pp. 2-6). Sample problems 1, and 4-6 in Pruess (1987) and verification problems 2 and 3 and validation problems 2 and 3 in Moridis and Pruess (1992) demonstrate the capability of TOUGH to model such heat flows in different geometries under varying initial and boundary conditions.
4. TOUGH was designed to calculate coupled transport of mass in two phases and heat (Pruess, 1987, pp. 2-6). Sample problems 4-6 in Pruess (1987) demonstrate the capability of TOUGH to simulate coupled transport of mass and heat in different geometrical settings.
5. The code has been developed to perform computations in porous and fractured media. (Pruess, 1987, pp. 3, 9). Sample problem 5 in Pruess (1987) demonstrates the capability of TOUGH to simulate porous and fractured media simultaneously in the same problem. Sample problems 2-4, 6 show the ability of the code to model flow and transport in porous media.

Pruess, Karsten, 1987. TOUGH User's Guide, Lawrence Berkeley Laboratory Report LBL-20700.

Moridis, George J., and Pruess, Karsten, 1992. TOUGH Simulations of Updegraff's Set of Fluid and Heat Flow Problems, Lawrence Berkeley Laboratory Report LBL-32611.

Use Qualification Report

Report prepared by: Karsten Pruess Date: March 8, 1993

Name of Code TOUGH Baseline ID: TOUGH 1.0.a

General Limitations and Conditions on Code Use:

The MA28 equation solver is subject to proprietary restrictions, and must not be used outside of the TOUGH program.

Limits on Parameter Values, if appropriate or if known:

Temperature range: $2°C \leq T \leq 360°C$

Pressure range: $0 \text{ bar} < P < 1000 \text{ bar}$

Limits on Boundary and Initial Conditions, if appropriate or if known:

The only known limitation is that time-dependent Dirichlet boundary conditions cannot be input directly. However, as shown in Moridis and Pruess (1992), they can be realized through appropriately chosen time-dependent sinks and sources.

Note: This form indicates that the above code has been qualified for use under the provisions of this SQAP, but not that it has been model validated for any user's application or for any site.

Review completed and approved:

[illegible] (Review Manager)

3/10/93 (Date)

Appendix B:

Sample Forms for STATEQ

Software Classification Form

Report prepared by: C. L. Carnahan Date: 3-5-93

Name of code: STATEQ Version: Aug. 1992

Purpose: To calculate static equilibrium distribution of aqueous species and either/equilibrate the aqueous species with selected solids or calculate saturation indices for selected solids.

Type

- [✓] Simulation code
- ☐ Data analysis software
- ☐ Auxiliary software
- ☐ Systems software

Origin

- [✓] New development software
- ☐ Acquired software
  - Source:
- ☐ Existing software

Review completed and approved:

Donald Mangold 3/5/93

(Review Manager) (Date)

Software Requirements Specification

Report prepared by: C. L. Carnahan Date: 3-5-93

Name of Code STATEQ Baseline ID: 1.0.a

See attached report.

Review completed and approved:

Donald Mangold
(Review Manager)

3/5/93
(Date)

Software Requirements Specification for STATEQ

This code will need to pass through all the stages of the development life cycle, including requirements definition, design specification, and implementation, as well as software procedure verification and software validation.

I. Functional Requirements

The name STATEQ signifies that this code is intended for geochemical static equilibrium studies. Therefore, the code must be capable of the following:

1. Calculate equilibrium concentrations of a given set of aqueous species from an input water analysis;
2. Equilibrate these aqueous species with (input) selected solids or calculate the saturation indices for them;
3. Incorporate formation of complexes and ion pairs in the aqueous phase, dissociation of water, reversible precipitation of stable solid phases, and oxidation-reduction (redox) reactions.

Basis aqueous species and solids must be input with identifiers and analytical concentrations. Equilibrium constants are to be calculated from input values of Gibbs free energies of formation. Activity coefficients are to be estimated by an extended Debye-Hückel formula.

Redox equations should be treated by two methods. One would be the "direct" method where the oxidation potential (Eh) is controlled directly by the chemical reactions included in the simulation. The other would be the "indirect" method where a hypothetical electron activity ("e^-") is defined to be a basis species.

II. Performance Requirements

The code should be written in standard FORTRAN 77 for portability and efficiency. It should be written in a modular style to enhance the maintainability and reliability of the code. Comments should be interspersed in the code for clear identification of its components to strengthen the assurance of its correctness, and for ease of understanding.

III. Interface Requirements

The code should be capable of stand-alone operation and also generating output to a general data file that could be utilized by another program.

Software Design Document

Report prepared by: C. L. Carnahan Date: 3-10-93

Name of Code STATEQ Baseline ID: 1.0.a

See attached report.

Review completed and approved:

Donald Mangold
(Review Manager)

3/11/93
(Date)

Software Design Document for STATEQ

I. Major Components of the Design Related to the Functional Requirements

The main functional requirements will be incorporated into the design as follows. A flow chart is attached.

A. Program Function and Flow

The MAIN program will control the sequence of ***input, calculation, and output*** by calling a sequence of subprograms where most of the work will be done (see flow chart).

B. Mathematical Problem to be Solved

See the user's guide (Carnahan, 1993) for the descriptions of the physical and chemical phenomena being modeled, the equations in the model, the notation used, and the assumptions and simplifications of the model equations. A set of algebraic equations is to be solved, some of which may be nonlinear. There are two kinds of equations in the set:

(1) Mass balance, depending on an input parameter, say INDEXI, that indicates which of the following will enter into the mass balance calculations:

(a) basis species only;

(b) basis species plus complexes containing the basis species;

(c) basis species plus complexes plus solids containing the basis species;

There is to be one equation for each (aqueous-phase) basis species; ***one*** equation may be replaced by a charge-balance equation that includes all aqueous-phase species.

(2) Balance equations for solids, comparing the current product of activities of the basis species in the solid to the theoretical solubility product of the solid.

C. Procedure to Solve the Equations

See the user's guide (Carnahan, 1993) for the descriptions of the solution techniques utilized. Newton-Raphson iteration shall be employed. This requires the following calculations:

(1) Calculate the residues of the equations described above using current values of the unknowns;

(2) Calculate individual elements of the Jacobian matrix;

(3) Solve the linearized matrix equation for the vector of corrections to the unknowns: concentrations of basis species and solids;

(4) Use corrections to calculate new values of the unknowns.

The calculations are iterated until the fractional change of each unknown is less than the input convergence criterion.

D. Auxiliary Calculations

The following are particular calculations done to support the main calculations described above:

(1) Eh is only computed once, if at all;

(2) Input free energies of formation are used to calculate formation constants and solubility products, done only once;

(3) Activity coefficients must be computed during each iteration of the solution procedure as described in Section 3 above, Procedure to Solve the Equations;

(4) Saturation indices are calculated only once, after convergence of the solution.

E. Ranges of Inputs and Outputs

The following are the ranges of inputs and corresponding outputs of the main variables used in the simulation:

(1) Temperature varies between 15^{o}C and 100^{o}C;

(2) pH varies between 0 and 14 (standard);

(3) Eh, if it is used at all, varies between -0.9 and 1.2 volts (standard);

(4) Ionic strength varies between 0 and 0.1.

II. Major Components of the Design Related to the Performance Requirements

The code should be written in standard FORTRAN 77 in a modular style with interspersed comments in the code for clear identification of its components. There should be provision for a title to identify a run. For control of the iterative solver, the maximum fractional change of the absolute value of any unknown parameter permitted for convergence should be specified by input value, and also the number of iterations between printouts when calculating the equilibrium distributions of chemical species should be specified.

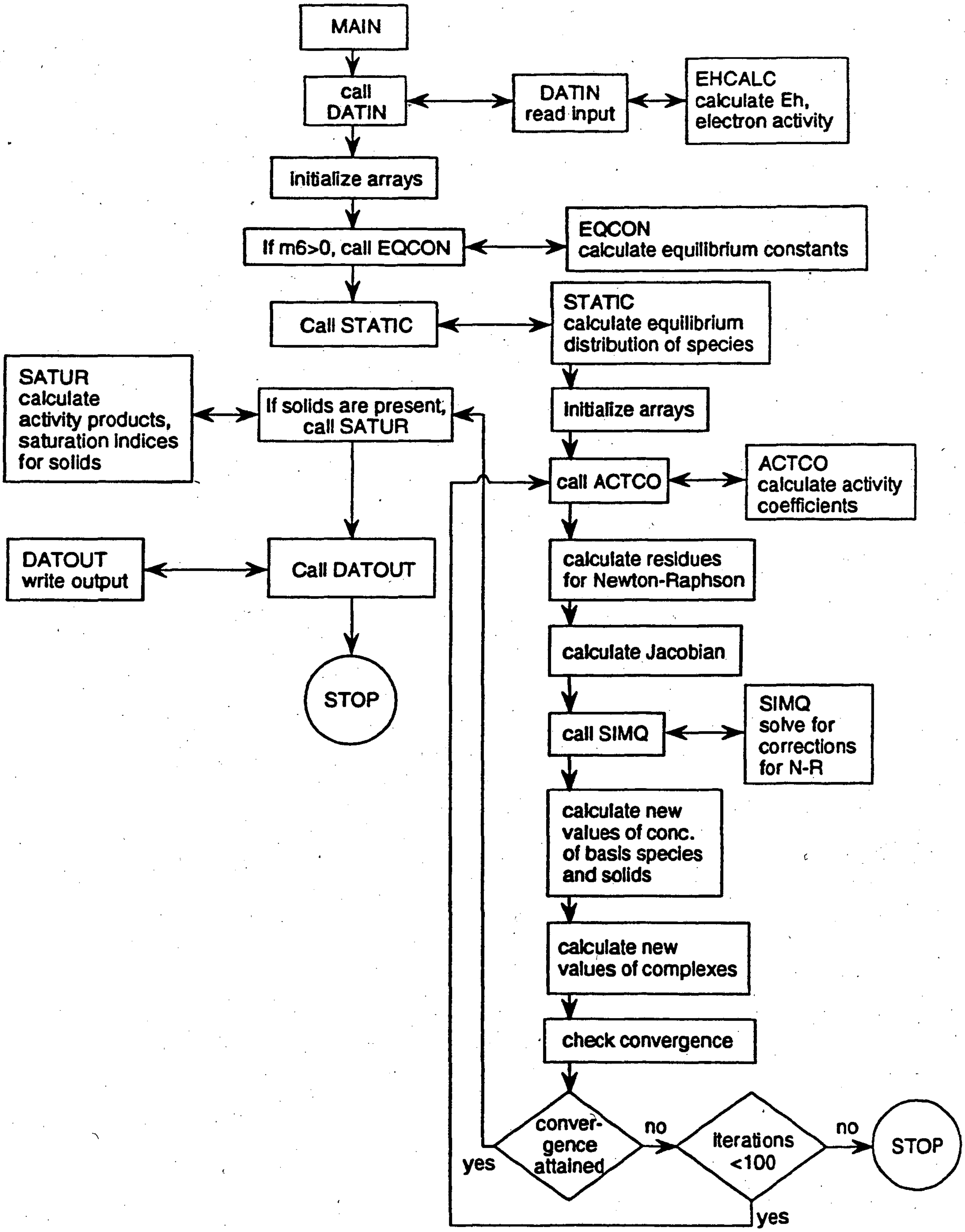
III. Major Components of the Design Related to the Interface Requirements

The code should be capable of stand-alone operation and generating output to a general data file that could be utilized by another program.

Reference

Carnahan, C. L., 1993. Computer Program STATEQ: A User's Guide, LBL-34007.

Flow Chart for STATEQ


MAIN
call DATIN
DATIN read input
EHCALC calculate Eh, electron activity
Initialize arrays
If m6>0, call EQCON
EQCON calculate equilibrium constants
Call STATIC
STATIC calculate equilibrium distribution of species
SATUR calculate activity products, saturation indices for solids
If solids are present, call SATUR
Initialize arrays
call ACTCO
ACTCO calculate activity coefficients
DATOUT write output
Call DATOUT
calculate residues for Newton-Raphson
calculate Jacobian
STOP
call SIMQ
SIMQ solve for corrections for N-R
calculate new values of conc. of basis species and solids
calculate new values of complexes
check convergence
convergence attained
yes
no
iterations <100
no
STOP
yes

LBL-34007
B-9

Computer Program STATEQ

A User's Manual

Chalon L. Carnahan
Earth Sciences Division
Lawrence Berkeley Laboratory
Berkeley, California 94720

April 1993

This work was supported by the Director, Office of Civilian Radioactive Waste Management, Office of Facilities Siting and Development, of the U. S. Department of Energy under Contract No. DE-AC03-76SF00098.

Software Procedure Verification Summary

Name of Code STATE Q Baseline ID: 1.0.a

Developer C. L. Carnahan

Requirements review and approval 3/12/93
(date)

Comments

Design review and approval 3/12/93
(date)

Comments

Implementation completion 3/16/93
(and ready for software validation) (date)

Comments

Review completed and approved:

Donald Mangold 3/16/93
(Review Manager) (Date)

Software Validation Test Plan

Report prepared by: C. L. Carnahan Date: 3-18-93

Name of Code STATEQ Baseline ID: 1.0.a

See attached report.

Review completed and approved:

[illegible] 3/19/93

(Review Manager) (Date)

Software Validation Test Plan for STATEQ

Overview

STATEQ should be tested or examined to ensure that it fulfills all the requirements set forth in the Software Requirements Specification (SRS) and Software Design Document (SDD).

In each of the following sections, the general means for software validation of the requirements of the SRS shall be testing of the code with inputs that exercise the specified functions. Some aspects will need inspection of the code listing. For checking the design of the code, the principal means of software validation shall be inspection of the code listing to verify that all the design requirements of the SDD are met.

The hardware and system software shall be the present LBL main computer system where the software resides. The code shall not perform any function that either by itself or in combination with other functions can degrade the entire computer system.

I. Test Plan for Functional Requirements

The code must be tested to be capable of the following:

1. Calculate equilibrium concentrations of a given set of aqueous species from an input water analysis;
2. Equilibrate these aqueous species with (input) selected solids or calculate the saturation indices for them;
3. Incorporate formation of complexes and ion pairs in the aqueous phase, dissociation of water, reversible precipitation of stable solid phases, and oxidation-reduction (redox) reactions.

Suitable input data shall be chosen to exercise the code for each of these requirements. The output shall be examined to assure that the calculations have been correctly performed.

Inspection of the code listing shall be made to ensure that basis aqueous species and solids are input with identifiers and concentrations, equilibrium constants are calculated from input values of Gibbs free energies of formation, and activity coefficients are estimated by an extended Debye-Hückel formula.

Inspection of the code listing shall also be made to ensure that redox equations are treated by two methods: the "direct" method where the oxidation potential (Eh) is controlled directly by the chemical reactions included in the simulation, and the "indirect" method where a hypothetical electron activity ("e^-") is defined to be a basis species.

The major components of the design in the SDD shall be checked by inspection of the code listing to ensure that the requirements are met for program function and flow, the mathematical problem to be solved, the procedure to solve the equations, and the auxiliary calculations. The ranges of inputs and outputs should be checked to ensure that they were correctly specified.

II. Test Plan for Performance Requirements

Inspection of the code listing shall be made to ensure that the code is written in standard FORTRAN 77 for portability and efficiency, and written in a modular style to enhance the maintainability and reliability of the code. It should also be checked whether there are interspersed comments in the code for clear identification of its components to strengthen the assurance of its correctness, and for ease of understanding.

The major components of the design in the SDD shall be checked by inspection of the code listing to ensure that the requirements are met for a title to identify a run, and control of the iterative solver is accomplished through input values for the maximum fractional change of the absolute value of any unknown parameter and input of the number of iterations between printouts when calculating the equilibrium distributions of chemical species.

III. Test Plan for Interface Requirements

Testing shall be performed to ensure that the code is capable of stand-alone operation and that it generates output to a general data file that could be utilized by another program.

Software Validation Test Results

Report prepared by: Donald Mangold Date: 3/25/93

Name of Code STATEQ Baseline ID: 1.0.a

See attached report.

Review completed and approved:

Donald Mangold	3/25/93
(Review Manager)	(Date)

Software Validation Test Results for STATEQ

Overview

STATEQ was tested and examined to ensure that it fulfills all the requirements set forth in the Software Requirements Specification (SRS) and Software Design Document (SDD).

In each of the following sections, the general means for software validation of the requirements of the SRS was testing of the code with inputs that exercise the specified functions. Some aspects had inspection of the code listing. For checking the design of the code, the principal means of software validation was inspection of the code listing to verify that all the design requirements of the SDD were met.

The hardware and system software were the present LBL main computer system where the software resides. The code did not perform any function that either by itself or in combination with other functions can degrade the entire computer system.

I. Test Plan for Functional Requirements

The code was tested to be capable of the following:

1. Calculate equilibrium concentrations of a given set of aqueous species from an input water analysis;
2. Equilibrate these aqueous species with (input) selected solids or calculate the saturation indices for them;
3. Incorporate formation of complexes and ion pairs in the aqueous phase, dissociation of water, reversible precipitation of stable solid phases, and oxidation-reduction (redox) reactions.

Suitable input data were chosen to exercise the code for each of these requirements. The output was examined to assure that the calculations had been correctly performed.

Inspection of the code listing was made to ensure that basis aqueous species and solids were input with identifiers and concentrations, equilibrium constants were calculated from input values of Gibbs free energies of formation, and activity coefficients were estimated by an extended Debye-Hückel formula.

Inspection of the code listing was also made to ensure that redox equations were treated by two methods: the "direct" method where the oxidation potential (Eh) is controlled directly by the chemical reactions included in the simulation, and the "indirect" method where a hypothetical electron activity ("e^-") is defined to be a basis species.

The major components of the design in the SDD were checked by inspection of the code listing to ensure that the requirements were met for program function and flow, the mathematical problem to be solved, the procedure to solve the equations, and the auxiliary calculations. The ranges of inputs and outputs were also checked to ensure that they were correctly specified.

II. Test Plan for Performance Requirements

Inspection of the code listing was made to ensure that the code is written in standard FORTRAN 77 for portability and efficiency, and written in a modular style to enhance the maintainability and reliability of the code. It was also checked as to whether there were interspersed comments in the code for clear identification of its components to strengthen the assurance of its correctness, and for ease of understanding.

The major component of the design in the SDD was checked by inspection of the code listing to ensure that the requirements were met for a title to identify a run, and control of the iterative solver was accomplished through input values for the maximum fractional change of the absolute value of any unknown parameter and input of the number of iterations between printouts when calculating the equilibrium distributions of chemical species.

III. Test Plan for Interface Requirements

Testing was performed to ensure that the code is capable of stand-alone operation and that it generates output to a general data file that could be utilized by another program.

Use Qualification Report

Report prepared by: C. L. Carnahan Date: 3 - 29 - 93

Name of Code STATEQ Baseline ID: 1.D.a

General Limitations and Conditions on Code Use:

(1) No transport calculations are performed (static calculations only)

(2) No kinetic calculations are performed (equilibrium calculations only)

Limits on Parameter Values, if appropriate or if known:

(as stated on Software Design Document, section E)

Temperature	$15°C \leq T \leq 100°C$
pH	$0 \leq pH \leq 14$
Eh (if used)	-0.9 volts $\leq Eh \leq 1.2$ volts

Limits on Boundary and Initial Conditions, if appropriate or if known:

Ionic strength $0 \leq I \leq 0.1 M$

Note: This form indicates that the above code has been qualified for use under the provisions of this SQAP, but not that it has been model validated for any user's application or for any site.

Review completed and approved:

[illegible] (Review Manager)

3/29/93 (Date)