

RECEIVED

JUN 05 1998

OSTI

PNNL-11880

UC-630

Pacific Northwest National Laboratory

Operated by Battelle for the
U.S. Department of Energy

An Approach to Ensuring Quality In Environmental Software

May 1998

Prepared for
Office of Research and Development
National Environmental Research Laboratory
U.S. Environmental Protection Agency
and
Office of Environmental Management
U.S. Department of Energy
and
Radiation Protection Division
Center for Risk Modeling and Emergency Response
U.S. Environmental Protection Agency
under Contract DE-AC06-76RLO 1830

Pacific Northwest National Laboratory
Richland, Washington 99352

PNNL-11880

MASTER
JM

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

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 Battelle Memorial Institute, nor any of their employees, makes any **warranty, expressed 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, or Battelle Memorial Institute. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE MEMORIAL INSTITUTE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC06-76RLO 1830

Printed in the United States of America

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37831;
prices available from (615) 576-8401.

Available to the public from the National Technical Information Service,
U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161



This document was printed on recycled paper.

An Approach to Ensuring Quality In Environmental Software

G. M. Gelston
R. E. Lundgren
J. P. McDonald
B. L. Hoopes

May 1998

Prepared for
Office of Research and Development
National Environmental Research Laboratory
U.S. Environmental Protection Agency
and
Office of Environmental Management
U.S. Department of Energy
and
Radiation Protection Division
Center for Risk Modeling and Emergency Response
U.S. Environmental Protection Agency
under Contract DE-AC06-76RLO 1830

Pacific Northwest National Laboratory
Richland, Washington 99352

Summary

Environmental software (see Appendix C for definitions of terms) is often used to determine impacts to the public, workers, and the environment from environmental contamination. It is vital, therefore, that the modeling results, and the software that provides them, be scientifically defensible and capable of withstanding the most rigorous of technical reviews. In other words, the control and assurance of quality is a critical factor for the project team that develops environmental software at the Pacific Northwest National Laboratory.

This document describes the philosophy, process, and activities that ensure a quality product throughout the life cycle of requirements analysis, design, programming, modification, testing, and implementation of environmental software. Quality is defined as the ability of the software to meet client needs. Meeting client needs starts with a shared understanding of how the software must perform and continues throughout the software life cycle through attention to details.

Environmental software developed by the project team is designed using an object-oriented approach. This software offers increased benefits, such as ease of maintenance and retention of the development and testing legacy of individual components, over traditional "hard wired" software. These benefits allow the design and testing of the models and future additions to be faster and less costly. This software is developed using a modular framework concept that allows a variety of models to work within a single construct.

This software has two parts: an overall system framework and a set of modules. Each module has up to three components: a user interface, a scientific model, and pre/post-processors. Each of these pieces has a different set of quality criteria associated with it. However, whatever form this software might take for a particular client, standard processes apply to protect the information from inappropriate use.

Figure S.1 outlines the environmental software development process with quality check points highlighted. The process shown was designed to be compatible with similar processes used by our clients. For example, our quality process compares favorably with that in the U.S. Environmental Protection Agency Directive 2182, *System Design and Development Guidance* (EPA 1997). It also compares favorably with the Office of Civilian Radioactive Waste Management's *Quality Assurance Requirements and Description, Supplement I, Software* (OCRWM 1995). Activities roughly equivalent across these processes are shown in Table S.1.

The information contained within this document can be applied to most environmental software to analyze risk in multiple environmental media, although in some cases client needs will require an even greater level of assurance. For specific projects, clients should refer to the proposal, statement of work, and/or project management plan for additional information on the detailed quality requirements and activities being planned.

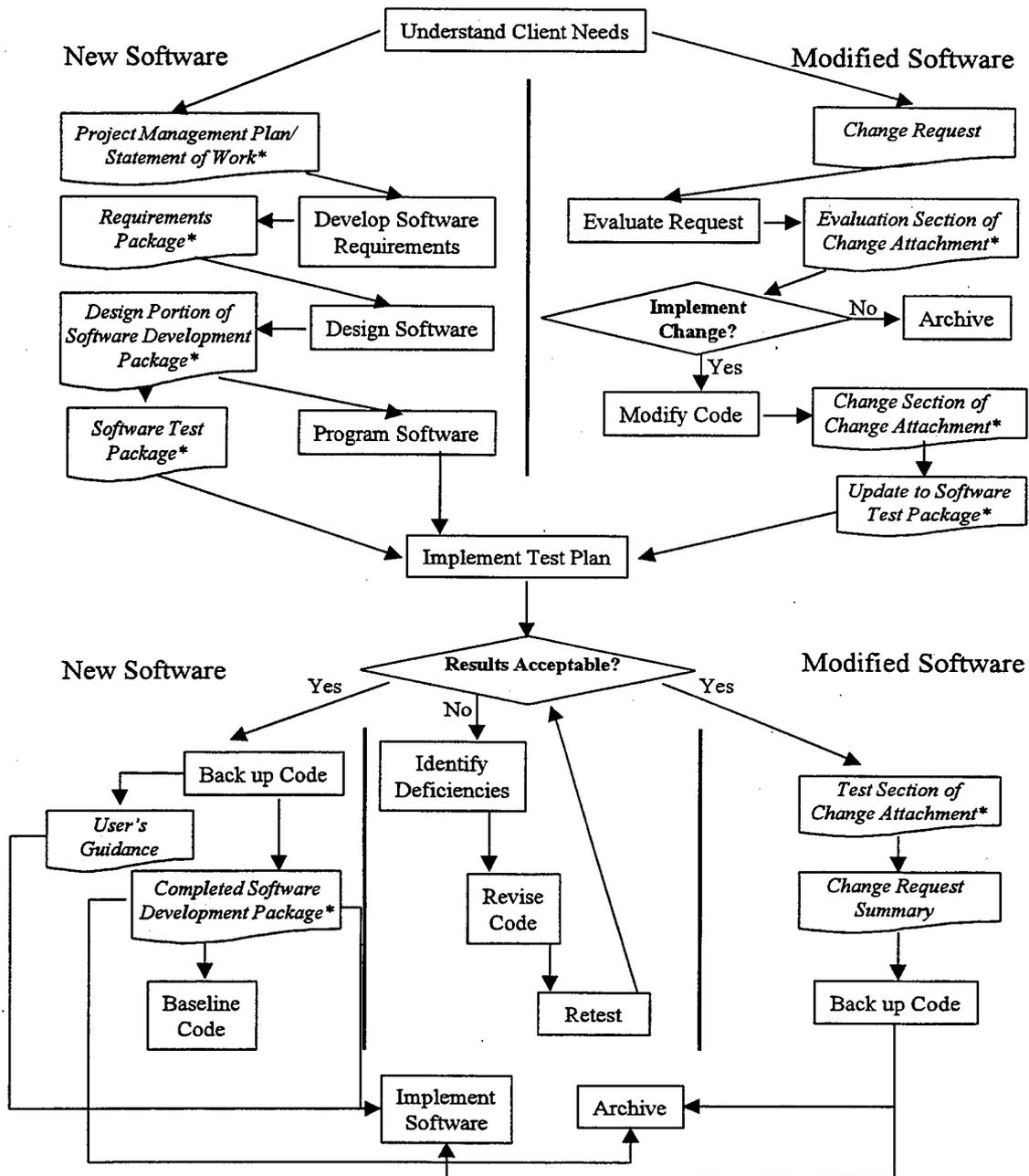


Figure S.1 Ensuring Quality in the Environmental Software Development Process

Table S.1 Relationship of Laboratory Environmental Software Development Process to the U.S. Environmental Protection Agency's Essential Elements of Information (EPA 1997)

| OCRWM Quality Assurance Requirement ^(a) | EPA Essential Element of Information ^(b) | Environmental Software Process Equivalent (Section) |
|---|---|--|
| | 4—System Implementation Plan | Project Management Plan or Statement of Work (see Appendix C for definitions of terms) (3.1.2) |
| I.2.5A Functional Requirements Information Documentation; I.2.5C Requirements and Design Documentation | 5—System Detailed Requirements Document | Requirements Package (3.1.2) |
| I.2.1 Software Life Cycles, Baselines (see Appendix C), and Controls | 6—Software Management Plan | Project Management Plan or Statement of Work (3.1.2) and this document |
| I.2.2 Software Verification ^(c) and Software Validation; I.2.4 Software Validation ^(d) | 7—Software Test and Acceptance Plan | Software Test Package (5.1) |
| I.2.3 Software Verification; I.2.5C Requirements and Design Information Documentation | 8—Software Design Document | Design Portion of Software Development Package (3.2.3) |
| I.2.6A Configuration Identification | | Completed Software Development Package (3.3) |
| I.2.6B Configuration Control; I.2.6C Configuration Status; I.2.7 Defect Reporting and Resolution ^(e) | 9—Software Maintenance Document | Modification Documentation (4.2) |
| | 10—Software Operations Document | User's Guidance and Training (3.4) |
| I.2.5B User Information Documentation | 11—Software User's Reference Guide | User's Guidance and Training (3.4) |
| | 12—System Integration Test Reports | Software Test Package (5.1) |

- (a) Note that OCRWM requirement I.2.8, Control of the Use of Software, is the responsibility of the OCRWM-related client.
- (b) Elements 1 through 3 are generally completed by clients in the U.S. Environmental Protection Agency before contract initiation with the project team.
- (c) Verification includes having software engineers informally test their code (see Appendix C) to ensure that it functions as required.
- (d) Validation includes testing by those other than the software engineers who developed the code to provide an independent confirmation that software functions as required.
- (e) Note that some changes requested by clients may not be made in the software unless funding has been allocated for such modifications.

References

Office of Civilian Radioactive Waste Management (OCRWM). 1995. *Quality Assurance Requirements and Description, Software*. U.S. Department of Energy, Washington, D.C.

U.S. Environmental Protection Agency (EPA). 1997. *System Design and Development Guidance*. EPA Directive Number 2182, Washington, D.C.

Contents

| | |
|---|-----|
| Summary | iii |
| 1.0 Introduction | 1.1 |
| 1.1 Purpose of This Document | 1.1 |
| 1.2 Philosophy of Software Development | 1.2 |
| 1.3 Scope of the Document | 1.3 |
| 1.4 Document Overview | 1.3 |
| 2.0 Software Approach | 2.1 |
| 2.1 Description of Software | 2.1 |
| 2.1.1 System Framework | 2.2 |
| 2.1.2 Module | 2.2 |
| 2.1.2.1 Model | 2.3 |
| 2.1.2.2 Module User Interface | 2.3 |
| 2.1.2.3 Module Pre/Post-Processors | 2.3 |
| 2.1.3 The User | 2.4 |
| 2.2 Information Security | 2.4 |
| 2.2.1 Applications Security | 2.4 |
| 2.2.2 Installation Security | 2.4 |
| 2.2.3 Information Management | 2.4 |
| 2.3 Software Safeguards and Sensitivity | 2.5 |
| 2.4 Potential Electronic Tracking System | 2.5 |
| 2.5 Performance Metrics | 2.5 |
| 2.5.1 System Framework | 2.5 |
| 2.5.2 Module User Interface | 2.6 |
| 2.5.3 Module Model | 2.6 |
| 2.5.4 Module Pre/Post-Processors | 2.7 |
| 3.0 Software Development | 3.1 |
| 3.1 Detailed Requirements Analysis | 3.1 |
| 3.1.1 Requirements Analysis | 3.1 |
| 3.1.2 Requirements Documentation | 3.1 |
| 3.2 Design | 3.3 |
| 3.2.1 Definition of Database and File Structure | 3.4 |
| 3.2.2 Confirmation | 3.4 |
| 3.2.3 Design Documentation | 3.4 |
| 3.2.4 Specification | 3.5 |
| 3.3 Programming and Programming Documentation | 3.5 |
| 3.4 Development of Software User's Guidance | 3.6 |
| 4.0 Software Modifications | 4.1 |
| 4.1 Performance Metrics Development | 4.1 |
| 4.1.1 Enhancements | 4.1 |
| 4.1.2 Errors and Bugs | 4.3 |
| 4.2 Modification Documentation | 4.3 |
| 4.2.1 Change Request | 4.3 |
| 4.2.2 Change Documentation | 4.4 |
| 4.2.3 Change Request Summary | 4.4 |
| 4.3 Design and Development | 4.4 |
| 5.0 Software Integration, Testing, and Evaluation | 5.1 |
| 5.1 New Software | 5.1 |

| | |
|--|-----|
| 5.2 Modified Software | 5.3 |
| 5.3 General Test Scenarios | 5.3 |
| 6.0 Software Implementation | 6.1 |
| 6.1 Technology Transfer | 6.1 |
| 6.1.1 Client Implementation Support | 6.1 |
| 6.1.2 Implementation Documentation | 6.1 |
| 6.1.3 User Training | 6.1 |
| 6.2 Software Operations and Maintenance | 6.2 |
| 7.0 References | 7.1 |
| Appendix A—Roles and Functions of Project Team Members | A.1 |
| Appendix B—Example Software Development and Modification Forms | B.1 |
| Appendix C—Glossary | C.1 |
| Appendix D—Outlines of Example Documents | D.1 |

Figures

| | |
|---|-----|
| Figure S.1 Ensuring Quality in the Environmental Software Development Process | iv |
| Figure 3.1 Process for Developing New Environmental Software | 3.2 |
| Figure 4.1 Process for Modifying Software | 4.2 |

Tables

| | |
|---|-----|
| Table S.1 Relationship of Laboratory Environmental Software Development Process to the U.S. Environmental Protection Agency's Essential Elements of Information | v |
| Table A.1. Key Activities During the Software Life Cycle and Team Members Responsible | A.5 |

1.0 Introduction

A variety of environmental management regulations requires computer models (see Appendix C) of varying sophistication for estimating the impacts of activities on humans and the environment. One example is environmental remediation and restoration activities under the Comprehensive Environmental Response, Compensation, and Liability Act (CERCLA) or Superfund. Another is the development, implementation (see Appendix C), and enforcement of regulations concerned with protecting human and ecological health from human-induced contamination. These types of regulations have led to a rapidly growing need for risk analysis software that takes a holistic approach to evaluating human health and ecological risks and hazards. Such software assesses impacts from a more comprehensive environmental systems perspective, cross-cutting various scientific disciplines. It also considers an increased number of interactions between contaminants, environmental media, and receptors (Whelan et al. 1997).

The Pacific Northwest National Laboratory (Pacific Northwest),^(a) has been in the forefront of developing such software for clients that span federal agencies, industry, and academia. This software is often used to determine impacts to the public, workers, and the environment from environmental contamination. The resulting information from the software is used in the context of important environmental decision making, affecting not only the regulatory agencies and potentially responsible parties, but the decision stakeholders as well. It is vital, therefore, that the modeling results, and the software that provides them, be scientifically defensible and capable of withstanding the most rigorous of technical reviews. In other words, the control and assurance of quality is a critical factor for the project team developing environmental software (see Appendix C) at Pacific Northwest.

1.1 Purpose of This Document

This document describes the philosophy, process, and activities that ensure a quality product in the requirements analysis, design, programming, modification, testing, and implementation (see Appendix C) of software to analyze risk in multiple environmental media. In most cases, the process described has been used for a number of years on dozens of projects, with similar positive results. The purpose of documenting the process at this time is to

- Provide a ready source of information for training new project staff
- Improve understanding of the process and thus acceptance of the final product
- Improve reliability of software by ensuring that all staff are following the same protocols
- Improve maintainability of the software by attention to careful documentation of modifications.

The information provided should help clients, general users (see Appendix C), and project team members understand the importance of ensuring quality in the software development life cycle.

A cornerstone of the process is adherence to laboratory standards at Pacific Northwest. The laboratory quality assurance standard states that

All staff shall document calculations, analyses, tests, and software required to substantiate results and processes used to develop products/solutions. Program managers shall manage assigned projects to a plan that appropriately documents

(a) Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under Contract DE-AC06-76RLO 1830.

deliverables, budget, schedule, management methods, organization and control systems (laboratory quality assurance standard, Standards Based Management System, 1997b).

In addition, the standard makes provisions for several levels of quality assurance, noting that

When a project meets the basic Battelle requirements (as provided in the Standards-Based Management System A-manuals and subject areas) or other project or activity documents sufficiently describe how customer requirements, drivers, and business, technical, or environment, safety and health risks are met, no additional quality assurance documentation is needed.

Accordingly, this document provides the standard quality assurance planning necessary for most projects to analyze requirements, design, program, modify, test, or implement software to analyze risk in multiple environmental media (some projects will require an even higher level of assurance based on the client's need).

Pacific Northwest also has a software development standard (laboratory computer software and database control standard, Standards Based Management System, 1997a) that embodies the quality standard and takes it several steps further. The standard requires that

Management shall promote the utilization of recognized system life-cycle management techniques to ensure quality and repeatable delivery of information systems and infrastructure services to both internal and external customers. Staff shall take reasonable actions to safeguard Pacific Northwest, Battelle, and client information assets, and computing and communications applications and resources against theft, loss, misuse and disruption.

Accordingly, this document describes how quality is managed throughout the software life cycle for environmental software (requirements analysis, design, programming, modification, testing, and implementation) and security measures commonly in place throughout the laboratory.

1.2 Philosophy of Software Development

We define quality as the ability of the software to meet client needs. Meeting client needs starts with a shared understanding of how the software must perform. It continues throughout the software life cycle through attention to details. For example, we use object-oriented programming constructs to control the flow of execution. This programming provides for well-defined interface points between modules and easier maintenance of the software. We also maintain a modular approach in source program design and coding to ensure compatibility, easier testing, and clear communication points. Finally, we practice good documentation in naming conventions, symbolic parameters, paragraphing, blocking, indentation of source code, specification of a single statement per line, and intelligent use of comments and error messages so that coding is easy to replicate, modify, and maintain. These standard practices allow us to develop high-quality software that satisfies our clients (see Appendix C for definitions of terms).

1.3 Scope of the Document

This document is meant to stand alone. The information contained within can be applied to most of the project team's software to analyze risk in multiple environmental media, although in some cases client needs will require an even greater level of assurance. For specific projects, clients should refer to the proposal, statement of work (see Appendix C), and/or project management plan for additional information on the detailed quality requirements and activities being planned.

In some cases, our clients ask us to apply the software we develop to a particular problem or need (for example, in estimating the risks of a major federal action for an environmental impact statement). Software application requires a different type of quality assurance process, one that considers software selection, data collection, software implementation, sensitivity and uncertainty analysis, and anchoring. This type of quality assurance process is not addressed in this document.

1.4 Document Overview

The following sections summarize quality control and assurance activities associated with a software life cycle. Section 2 provides a historical perspective of such software development as well as a general description of the software and certain applications and information security measures taken across projects. Section 3 describes the life-cycle process for the development of new software. Section 4 provides similar information for modifications to existing software. Section 5 details the test process used for both new and modified software. Section 6 describes the support provided to transfer the technology and implement the software at the client's organization. Section 7 provides references cited elsewhere in this document. Appendix A describes the roles and functions of project team members. Appendix B provides example forms used to track design, programming, and modification of environmental software. Appendix C provides a glossary of specialized terms used in this document. Appendix D provides example outlines for documentation packages (see Appendix C) described in this document.

2.0 Software Approach

The project team uses an object-oriented approach to design environmental software. This software offers increased benefits over those of traditional "hard-wired" (see Appendix C) software. Over the past 35 years, this traditional environmental software has been developed for specific media (soil, groundwater, surface water, air, etc.) in an effort to understand and predict environmental phenomena. Such software is still being developed today. The evolution of this software has followed a logical progression:

- In 1959, the Stanford Watershed Model represented one of the first "integrated" models, as it linked multiple processes by simulating the land-phase of the hydrologic cycle for the entire watershed.
- In 1969, Oak Ridge National Laboratory presented the Unified Transport Approach, which coupled (i.e., "hard-wired") detailed numerical models, describing individual environmental media. This model did not progress into general use because 1) the models were difficult to understand, operate, modify, and maintain, 2) data to operate the models were generally unavailable, and, most importantly, 3) computer power to drive the software was lacking at the time.
- In 1984, the introduction of desk-top computing allowed the first fully coupled sequential multiple media model (see Appendix C), which accounted for temporally and spatially varying contamination within designated media. Each medium-specific model was "hard-wired" into the software, so replacing these components (see Appendix C) was not easy.
- Around 1990, the development of large multi-purpose frameworks (see Appendix C) began, which "hard-wired" a suite of codes together and investigated not just the distribution of contaminants in the environment, but the relationships between a suite of issues deemed valuable (e.g., regulatory criteria, data quality objectives, regulatory processes, etc.).

Unfortunately, one of the drawbacks of this "hard-wired" software was that in incorporating individual components, the legacy of development and testing for that component was compromised. Even when these components could be incorporated intact, modifications were often necessary in other components. Therefore, the modification and maintenance of this software were costly and time-consuming. A clear solution was to move toward a more object-oriented design, which is easier to maintain, retains the development and testing legacy of individual components, and thus makes design and testing of the software and future additions faster and less costly.

2.1 Description of Software

The project team develops software for risk analysis in multiple environmental media within a modular framework that allows users the flexibility to construct, combine, and couple attributes that meet their specific needs. This framework allows a variety of models to work within a single construct. This software has two parts: an overall system framework and a set of modules. Each module has up to three components: a user interface, a scientific model, and a pre/post-processor (see Appendix C). Each of these pieces has a different set of quality criteria associated with it.

2.1.1 System Framework

The system framework typically consists of a set of modules, an associated framework user interface, and data exchange specifications. The purpose of a system framework is to

- minimize data-exchange requirements between modules of the system framework
- allow relatively easy inclusion of additional modules and models
- allow for unlimited access to data
- address linkage concerns for a variety of models.

The system framework typically includes an interface to enable the user to access these capabilities easily.

Depending on client needs, the framework may accommodate various levels of detail (i.e., resolution) of the models and the scale of assessment (e.g., medium-specific, watershed, regional, and global). It may also access a number of site- or installation-specific and national databases such as population census data maintained by the U.S. Environmental Protection Agency Office of Air Quality Planning and Standards and surface water gauging station data are maintained by the U.S. Geological Survey.

The project manager assembles a team of researchers for each project to develop and maintain the system framework. Team roles (which are described in more detail in Appendix A) include the following:

- project manager—provides client interface and leadership; ensures compliance with laboratory policies and procedures for administration, scientific merit, and software development
- integration leader—ensures appropriate quality in meeting client needs across all pieces of the software; provides as well as project management
- lead software engineer—designs and develops software to meet client needs
- subject-matter expert—ensures software has strong scientific basis in discipline being modeled (environmental science, uncertainty analysis, etc.)
- tester—ensures software developed meets client needs and can be implemented.

Depending on the size and complexity of the project, there may also be additional software engineers, a task leader who oversees the development of a particular piece of software, and a technical reviewer who ensures the scientific integrity of the software.

2.1.2 Module

Each module potentially contains three components: the user interface, the scientific model, and, for that software which incorporates legacy models, pre- and/or post-processors. Examples of modules include source-term releases, vadose-zone transport, saturated-zone transport, surface-water transport, air transport, exposure-pathway analysis, dose estimates, health impacts, and sensitivity/uncertainty support tools.

The project manager assembles a team of researchers to develop each module, as noted above for system framework development. However, once a module has been developed, a different team ensures that it is maintained appropriately. Because of Pacific Northwest's extensive experience in developing environmental software, we have readily available a number of our modules. Sometimes these can be used with no modifications to solve new problems. In

other cases, these can be used with minimal modifications. A module manager, with the assistance of a code custodian and subject-matter expert, manages these modules across projects. The module manager ensures that the software continues to serve the application for which it was originally intended as well as the new application. The software engineer ensures that changes to the code are properly documented and traceable. The subject-matter expert ensures that changes to the software do not jeopardize the scientific basis. Additional information on the roles and functions of each of these team members can be found in Appendix A.

2.1.2.1 Model

The model is the set of scientific calculations that defines a particular module. Several models have been developed over the past 10 years by researchers focusing on developing fully integrated, physics-based, intermedia modules that allow a more transparent connection between individual medium-specific models. The grouping of these physics-based models takes a holistic approach to environmental assessment of potential contaminant impacts as they simulate

- the release of contaminants into the environment
- migration and fate through various environmental media (i.e., groundwater, surface water, air, and overland surfaces)
- resultant exposures and impacts.

These models also provide support tools such as sensitivity, uncertainty, graphical interface systems, and results display.

The overall scope of these models generally includes evaluation of on- and off-site impacts from active and inactive sites involving both chemical and radioactive wastes. Although differing in their individual scopes, these multiple media models tend to be "analytical" in nature (e.g., mainly compartmental, analytical, semi-analytical, and empirical algorithms). Other numerical or structured-value models can be used within this holistic approach or as an outside model.

2.1.2.2 Module User Interface (see Appendix C)

The main purpose of the user interface to a module is to make it easy to collect the data necessary to run the model. The user interface also often provides online help to the user, lists storage options for collected data, provides flexible unit inputs (see Appendix C), and serves other user-support functions.

2.1.2.3 Module Pre/Post-Processors

As mentioned earlier, it saves time and cost if models can be integrated into a system framework intact. Legacy software that has been tested and reviewed can be preserved and integrated by adding pre- and/or post-processors to the module. These processors transfer reorganized data into the specified format of the overall framework, thus allowing the inclusion of modules that were initially created for a medium-specific analysis to be used in this more holistic approach to multiple media assessments. Whether a pre/post-processor is used depends on the needs of the scientific model and the specification of the framework. Models that have been created or modified with the specifications predefined will likely not need pre/post-processors before integration.

2.1.3 The User

The anticipated user of this overall framework and its modules is expected to have some environmental science knowledge and to be familiar with the computer platform on which the application is housed. In addition, completion of a training session or online tutorial is also recommended for potential users. Although the user interfaces are intended to be intuitive and user friendly, this may not seem the case for specific users; thus, a training session or completion of a tutorial can help address common problems encountered by new users.

2.2 Information Security

Whatever form our software might take for a particular client, standard processes apply to protect the information from inappropriate use. These processes include application security, installation security, and the confidentiality, integrity, and availability of information.

2.2.1 Applications Security

All computer systems and related software at Pacific Northwest are used for official business and activities sanctioned by management. Protection systems are in place to prevent sabotage or damage by viruses. Physical security measures include the following:

- Staff know the people who have routine access to the area.
- Staff supervise the use and maintenance of their systems.
- All systems are in managed buildings (access through security checkpoints).
- Approval is required before non-laboratory personnel are granted access to computing resources.
- Backed up copies of software and files are stored in other locations (see information management below).

To protect against viruses, virus scans are performed monthly on all networked computers and at least quarterly on standalone models. In addition, all disks and files from unknown or questionable systems are scanned before use. Pacific Northwest uses a nationally recognized virus scanning program capable of identifying most forms of viruses including the newer macro-viruses.

2.2.2 Installation Security

Most risk-analysis software developed by the project team comes in installation disk sets. Users generally install the disks into the same directory or folder. To ensure ease of installation, all software is installed using general standard installation procedures. These procedures prompt users through the installation process.

2.2.3 Information Management

All software covered by this approach is designed to meet client needs; each client receives specific information and software to this end. However, in all cases, the master source codes remain the property of Pacific Northwest. Electronic copies are backed up and kept in at least two different buildings as well as on a networked fixed disk; a baseline (see Appendix C)

printout is also kept separately. Both the code custodian and the integration leader keep a completed form detailing all locations for the respective pieces.

2.3 Software Safeguards and Sensitivity

Our risk analysis software is generally used to estimate the impacts to human or ecological health from contaminant releases to the environment through several pathways of exposure. This information is generally used to hypothesize impacts from new contaminant sources or changes to existing sources (such as effects from environmental remediation and restoration). The information can also be used to monitor compliance with environmental regulations. Clients may use it for a single activity or to analyze a full suite of activities spanning multiple installations and locations.

Because the ways in which the software can be used differ between clients, the software's sensitivity can also vary widely. Therefore, it is the client's responsibility to determine appropriate safeguards and security necessary for their particular use. Project staff will discuss this need with the client early in the planning process so that client requirements can be built into software development.

2.4 Potential Electronic Tracking System

The processes detailed in this report are tracked via an electronic spreadsheet program. However, an increasing number of automated software quality assurance tools designed for Internet, Intranet, and Web-based use may save time and money. Change request (see Appendix C) tools introduce a consistency of communication that makes data gathering a simple, painless process which encourages people to report defects. Enhancement requests, defect reports, and changes can be easily managed from initial incident through resolution and testing. Version-control management tools can reduce the risk of change by enforcing and recording the change process. The right change request and version control tools can save time, ease software maintenance, and coordinate the work of multiple team members. The project team is investigating the utility and feasibility of incorporating one or more of these tools to maximize the quality assurance process for clients.

2.5 Performance Metrics

The performance metrics for this approach consider eight areas: compatibility, completeness, consistency, correctness, ability to be modified, robustness, understandability, and testability. Many of these areas are generally addressed in our standard use of object-oriented design. The following are examples of the performance metrics specific to the four pieces of software described in Section 2.1 (framework, module user interface, model, and pre/post-processors).

2.5.1 System Framework

Answering yes to the following questions indicates that the system framework will perform in accordance with quality expectations:

- Does the system framework design have a specification for data transfer between modules?
- Do the interface requirements ensure that modules will be compatible?

- Does the requirements package include all requirements defined in the project proposal (as documented in the project management plan or statement of work)?
- Do all documentation packages use standard terminology and definitions throughout?
- Are the requirements compatible with the hardware and software that will be used in the operational environments?
- Do interface requirements define the required responses to potential types of errors and failure modes identified?
- Is there justification for design/implementation constraints?
- Are requirements organized to allow for modifications?
- Are there requirements addressing fault tolerances and graceful degradation so that an execution of the software does not impact computer performance?
- Does the interface have guidance to aid the user?
- Does documentation contain only necessary implementation details?
- Are requirements clear and specific enough to be the basis for design guidance and functional tests?
- Does documentation differentiate between requirements and other information provided?
- Is there a test defined for each general requirement?
- Does the system framework perform acceptably for all the identified tests?

2.5.2 Module User Interface

Answering yes to the following questions indicates that the module user interface will perform in accordance with quality expectations:

- Does the requirements package include all requirements defined in the project proposal (as documented in the project management plan or statement of work)?
- Are the module user-interface requirements consistent with framework-specification requirements?
- Does documentation use standard terminology and definitions throughout?
- Are requirements compatible with the hardware and software that will be used in the operational environment?
- Do interface requirements define the required responses to potential types of errors and failure modes identified?
- Is there justification for design/implementation constraints?
- Are requirements organized to allow for modifications?
- Are there requirements addressing fault tolerances and graceful degradation so that an execution of the software does not impact computer performance?
- Does the interface have guidance to aid the user?
- Does documentation contain only necessary implementation details?
- Are requirements clear and specific enough to be the basis for design guidance and functional tests?
- Does documentation differentiate between requirements and other information provided?
- Is there a test defined for each general requirement?
- Does the module user interface perform acceptably for all identified tests?

2.5.3 Module Model

Answering yes to the following questions indicates that the module model will perform in accordance with quality expectations:

- Does the requirements package include all requirements defined in the project proposal (as documented in the project management plan or statement of work)?
- Are formulations free of contradictions?
- Are the formulations complete?
- Are the specified algorithms and numerical techniques compatible?
- Are requirements compatible with the hardware and software that will be used in the operational environment?
- Are algorithms and regulations supported by scientific or other appropriate literature?
- Is there justification for design/implementation constraints?
- Are requirements organized so as to allow for modifications?
- Are there requirements addressing fault tolerances and graceful degradation so that an execution of the software does not impact computer performance?
- Does documentation contain only necessary implementation details?
- Are requirements clear and specific enough to be the basis for design guidance and functional tests?
- Does documentation differentiate between requirements and other information provided?
- Are mathematical functions defined in documentation using notation with well-defined syntax and semantics?
- Is there a test defined for each general requirement?
- Does the model perform acceptably for all the identified tests?

2.5.4 Module Pre/Post-Processors

Answering yes to the following questions indicates that the module pre/post-processors will perform in accordance with quality expectations:

- Do processor requirements enable the external model components to be integrated into the framework system?
- Are pre/post-processors requirements consistent with framework-specification requirements?
- Are requirements organized to allow for modifications?
- Is there a test defined for each processor requirement?
- Do the module pre/post-processors perform acceptably for all the identified tests?

3.0 Software Development

The project team follows a specific process (Figure 3.1), honed by years of experience with a variety of clients, in developing new software. Client needs, programming standards and languages, and available development tools influence this development. In general, this process entails understanding what the client requires the software to do, designing the software to meet those requirements, and programming to respond to the design. Testing occurs at all steps in this process. For example, requirements and design are tested through reviews by team members and the client. The actual software developed is tested by running and evaluating test cases (see Section 5.0). Appendix A describes the responsibilities of various roles discussed below.

3.1 Detailed Requirements Analysis

Each project starts by defining the client's needs. What problem must be solved? What kinds of information are needed? Who are the ultimate users and how can the software best meet their needs? Answers to these questions are prerequisites to a definition of the specific requirements for software.

3.1.1 Requirements Analysis

The requirements analysis is based on communication with the client and historical use of software by the project team. The project team has years of experience in analyzing environmental issues and has developed significant tools for use in understanding these issues. This experience, along with understanding the specific needs of the client, is the basis for the requirements analysis, which is documented in the proposal or statement of work for the client.

An important part of the requirements analysis is to define the functional components of hardware and software. What hardware systems are being used by the client? Are there any software packages, such as standard packages for word processing or database management, that must be considered at the client's organization? These components are determined by the client's hardware environment and the environments of legacy software being incorporated.

3.1.2 Requirements Documentation

The information developed during the requirements-analysis phase of the project is gathered into the requirements package. This requirements description includes two levels of detail, general requirements and specific requirements. Many of the general requirements are described in project documentation, such as the project management plan with task descriptions and/or statement of work for the project. These documents also list the deliverables specific to each task, such as requirements analysis, user's guidance, and testing approach. To ensure accuracy and client understanding and support, the statement of work is approved by the client before the initiation of work. (The project management plan or statement of work contains similar information as that in the U.S. Environmental Protection Agency's "System Implementation Plan" and "Software Management Plan," Essential Elements of Information 4 and 6, respectively, in EPA 1997.)

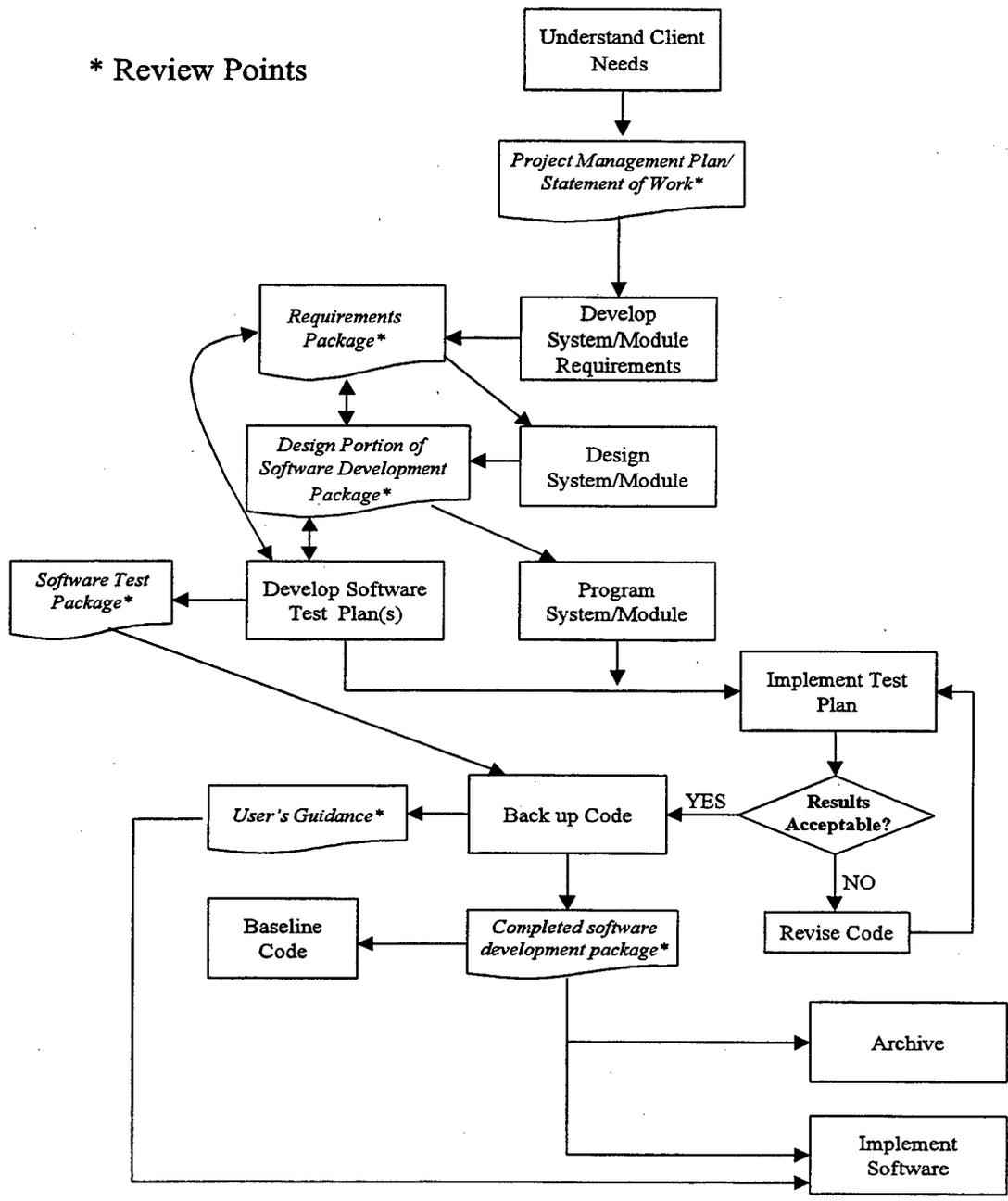


Figure 3.1. Process for Developing New Environmental Software

The information in the project management plan should, at a minimum, answer the following questions:

- Which capabilities have been discussed with the client (which are they expecting the project team to use on this project)? What additional capabilities are necessary to produce a quality product?
- What specific restrictions have been noted?
- What potential difficulties have been identified?
- What compatibilities are necessary for usability (confirm compatibility with client's systems)?
- Who are the project team members?
- What are the expected deliverables for the task?

In addition to the statement of work and task descriptions, additional general requirements can be documented and included in the requirements package. The requirements package should be sufficiently detailed to be used as the foundation for design and testing. It should list requirements in general categories, breaking each category down into sufficient detail that all team members will clearly understand what is expected as an outcome of the software. (The requirements package contains similar information as that in EPA's "System Detailed Requirements Document," Essential Element of Information 5 in EPA 1997. The requirements package also contains similar information as that in the Office of Civilian Radioactive Waste Management's Quality Assurance Requirement I.2.5A, "Functional Requirements Information Documentation" and I.2.5C, "Requirements and Design Documentation" in OCRWM 1995). The following information is discussed in a requirements package:

- purpose of the software
- structure of the software (possibly through the use of a flowchart)
- hardware and software requirements, such as memory, processor speed, and system platform
- input requirements for each component
- output (see Appendix C) requirements for each component
- scientific basis for the component
- assumptions on which the science was based
- limitations of the various components.

The package may consist of an annotated outline or work breakdown structure, compilation of materials, or a publishable report, depending on client needs. An example outline of a requirements package is shown in Appendix D.

The subject-matter expert leads team members in determining what the requirements are for the module or system framework. The client, project manager, tester, and integration leader review the requirements package before finalizing to ensure that client needs are met and the software can be developed, tested, and implemented.

3.2 Design

Software design is the process of taking the information in the requirements package and translating it into software. The lead software engineer, who may have assistance from other software engineers as well as a subject-matter expert, leads the process. Design includes definition of database and file structure, initial and final design, and documentation.

3.2.1 Definition of Database and File Structure

Before code is designed, the lead software engineer must determine the appropriate databases to be used and the file structures for input and output from them. Module file structure should be consistent with the system framework's file specification format. Pre/post-processors may be used to help convert legacy code file formats not already meeting those specifications. All file formats should be designed to ensure readability and compatibility with most spreadsheet programs and to incorporate necessary information to communicate the use and purpose of each file.

3.2.2 Confirmation

The lead software engineer, with help from other software engineers as appropriate, determines what kinds of software pieces and coding must be developed to meet the requirements. Every requirement level and sub-level is addressed. These design elements are specified sufficiently clearly to allow understanding by all team members.

Once the initial design has been determined, the subject-matter expert and lead software engineer confirm relationships between the requirements and design. At this time, the subject-matter expert and lead software engineer may decide on a ranking of both the requirements and related design elements to ensure that the development process addresses the more critical items before others.

3.2.3 Design Documentation

Both design and programming activities are captured in a software development package (see Appendix C), which can include one or more of the following:

- a matrix that lists software pieces or processes next to requirements (traceability matrix)
- a database that correlates requirements with design elements
- flowcharts and block diagrams to describe the linkages and solution strategy.

Regardless of whether the software being documented is a system framework or a module, the design portion of the software development package identifies the type of code (new, replacement, upgrade) and the members of the development team. It provides a generic description of the code, often in the form of a task description or flowchart showing modules and linkages. It also lists such things as

- specifications, including communication file formats
- file format descriptions for those files that do not meet framework system-level specifications
- general statements describing the functionality of each module
- logic diagrams showing the design of the system framework-level codes
- necessary help information
- ways to ensure a consistent look and feel with related interfaces.

The design documentation also describes mathematical formulations on which the software will be based. This description should encompass all scientific algorithms used in the software and describe variables, defaults, and how the algorithms will be incorporated. In addition, the

description should include confirmation that parameters which must be derived from one another are actually derived from one another or at least maintain their direct relationships.

Depending on the sensitivity or complexity of the formulations noted above, these formulations can be reviewed internally and/or externally, with signatures and comments noted and addressed in the design portion of the software development package. Reviewers can be other subject-matter experts or software engineers, or a representative of the client. The design portion of the software development package also addresses the need to develop pre/post-processors to enable the module to function within the overall system framework. When the design portion of the software development package is completed, the design is reviewed by the subject-matter expert, tester, and project manager and task leader if there is one. The design is approved by the subject-matter expert and lead software engineer.

The software development package is helpful throughout the process because it captures the lead software engineer's understanding of the requirements and provides an opportunity for internal and external reviews of the design. In addition, the tester reviews the design to ensure that tests can be developed appropriately. (The software development package contains information similar to that in the EPA's "Software Design Document," Essential Element of Information 8 in EPA 1997. It also contains similar information to that in OCRWM's Quality Assurance Requirement, Sections I.2.3, "Software Verification;" I.2.5C, "Requirements and Design Information Documentation;" and I.2.6A, "Configuration Identification"). A form commonly used for the software development package is shown in Appendix B. An outline for a design document, which is used for more complex software, is shown in Appendix D.

3.2.4 Specification

After the design has been reviewed, the lead software engineer addresses the finer points of the design for the software development package. The process of evaluating the current set of design elements by asking the question "how will each of these elements be addressed?" is often used to break down the design into workable sections that can then be assigned to other software engineers.

3.3 Programming and Programming Documentation

Programming begins when the design is complete and has been reviewed by the subject-matter expert, tester, project manager, and task leader and approved by the subject-matter expert. Software engineers use object-oriented programming techniques to ensure a quality product. When programming is completed, the lead software engineer completes the software development package. Then the subject-matter expert, task leader, and project manager review the package. When reviews are complete, the project manager works with appropriate line managers to select a module/system framework manager and code custodian for future modifications to the software.

Regardless of whether the software being documented is a system framework or a module, the information in the programming portion of the completed software development package includes a baseline hard copy of the source code listing as well as a diskette copy labeled with software name, version, developer names, and date. The diskette includes the source codes, any executable files, and any "readme" files with special instructions to users. The package also documents the computer programming language used and any other additional languages used. Changes to the components of the software after the software development package is complete require the additional signature of the module/system framework manager (see Section 4.0).

An example form used to document programming activities for the software development package is shown in Appendix B. An outline for a specifications document, which is sometimes used to describe programming for more complex software, is included in Appendix D.

3.4 Development of Software User's Guidance

Software user's guidance is developed for each module or system framework. This user's guidance, often in the form of online help, is generally an associated real-time system using hypertext language. However, this type of help information is also made available for printing in hard copy form to function as a traditional user's guide. (The user's guidance, along with the training discussed in Section 6.0, contains similar information as that in EPA's "Software Operations Document" and "Software User's Reference Guide," Essential Elements of Information 10 and 11 in EPA 1997. It also contains information similar to that in OCRWM's Quality Assurance Requirement, Section I.2.5B, "User Information Documentation," in OCRWM 1995.)

In addition, online tutorials may be developed for specified software. These tutorials serve as a supplement or replacement to the traditional client training arranged to aid in software implementation. Tutorials communicate the information needed to operate the software correctly. Information is provided on the operation of the user interface and underlying models and their appropriate use. Whether an online tutorial is developed is determined by the client and documented in the project management plan or statement of work.

4.0 Software Modifications

Over the years, the Pacific Northwest has developed a variety of software for risk analysis in multiple environmental media, such as the Multimedia Environmental Pollutant Analysis System (MEPAS), Remedial Action Assessment System (RAAS), Generation II of the Hanford Environmental Dosimetry Codes (GENII), and others. While some client needs necessitate the development of entirely new software, often modifications to existing software are more cost-effective and useful. Modifications are influenced by programming language constraints, detailed user requirements, data requirements, and the physical environment. The approach to modifications is detailed below (see Figure 4.1). Descriptions of the roles and responsibilities of key project staff can be found in Appendix A. Example forms used in software modification tracking can be found in Appendix B.

4.1 Performance Metrics Development

As mentioned in Section 2.5, the performance metrics for our environmental software consider eight areas: compatibility, completeness, consistency, correctness, ability to be modified, robustness, understandability, and testability. Many of these areas are addressed in our standard approach of object-oriented design and thus have been addressed in the development of the original software. This software may be modified for one of two reasons: either a client or other user has suggested an enhancement they would like to purchase, or the project team or user has identified an error or "bug" (see Appendix C) in the software. The following are examples of performance metrics specific to these types of modifications.

4.1.1 Enhancements

An enhancement might be made to the system framework or one of the modules. When enhancing a system framework, answering yes to the following questions indicates that the software will perform in accordance with quality expectations:

- Were modifications evaluated for appropriateness and feasibility?
- Were modifications to data communication specifications documented?
- Were modifications approved by the subject-matter expert and system framework manager?
- Were modifications documented in such a way as to ensure that the process could be reproduced?
- Did baseline test cases reproduce expected results?

When enhancing a module, answering yes to the following questions indicates that the module will perform in accordance with quality expectations:

- Were modifications evaluated for appropriateness and feasibility?
- Were modifications to model formulations documented?
- Were modifications approved by the subject-matter expert and module manager?
- Were modifications documented in such a way as to ensure that the process could be reproduced?
- Did baseline test cases reproduce expected results?
- Were system and other module interconnectivity needs considered?

*** Review Points**

Assumes software is right approach and exact software is not available

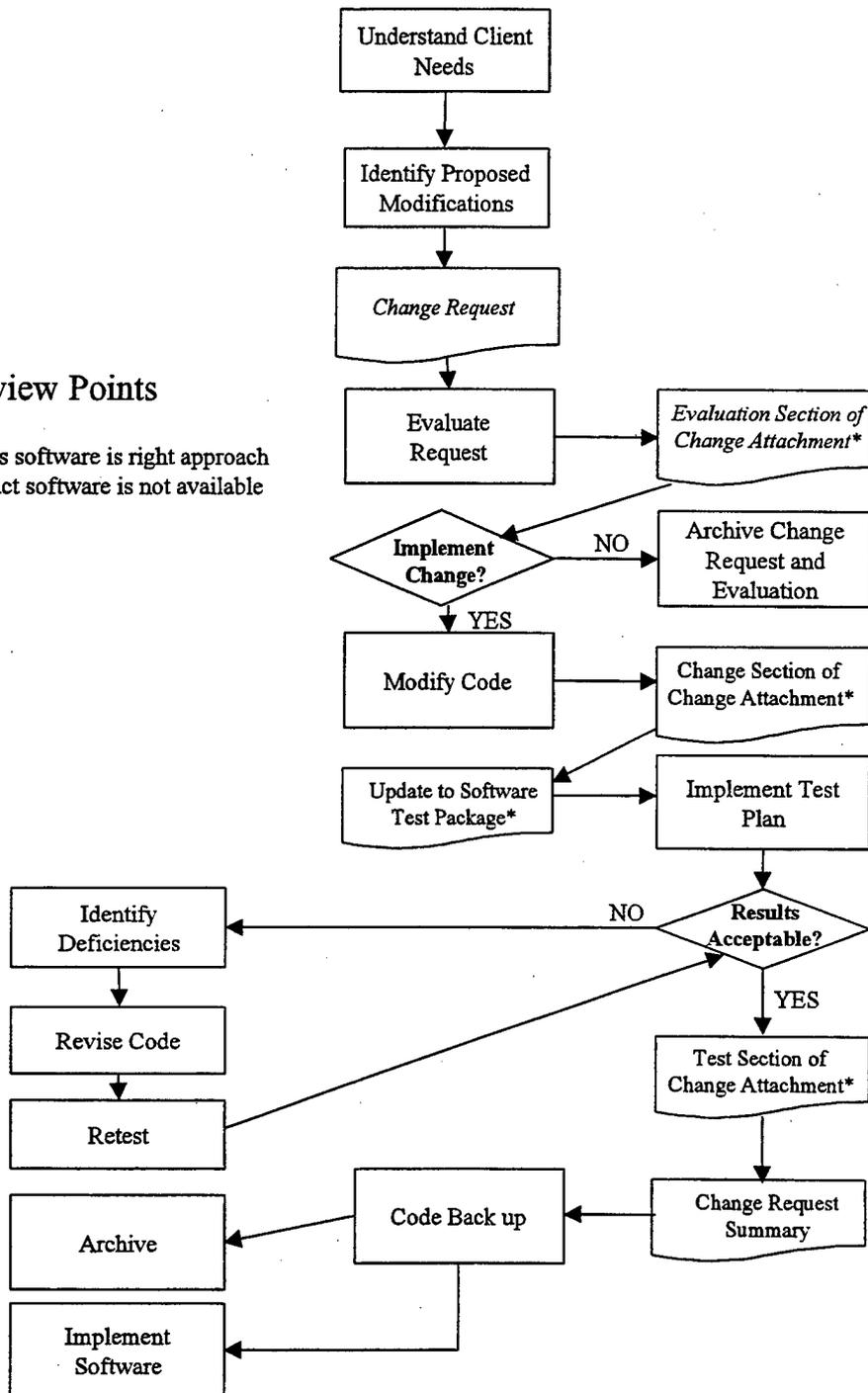


Figure 4.1 Process for Modifying Software

4.1.2 Errors and Bugs

Once a potential problem has been identified, the project team attempts to reproduce the problem (i.e., determine the steps that led to the error message or other problem). From this information, the code custodian identifies a potential way to fix the problem; the proposed method is approved by the cognizant subject-matter expert. When implementing the proposed change, answering yes to the following questions indicates that the problem has been solved:

- Was the change evaluated for appropriateness and effect on other segments of the code?
- Was the change approved by the subject area expert and module/system framework manager?
- Was the change documented in such a way as to ensure that the steps taken can be reproduced?
- Were baseline test cases reproduced with the expected effect on results?

4.2 Modification Documentation

Once software has been developed to a baseline, modifications must be planned carefully to ensure minimal impact on existing users. Key to this is tracking the requested changes and their expected impacts on results. (The three pieces described below—change request, change documentation, and change request summary [see Appendix C]—contain information similar to that in EPA's "Software Maintenance Document," Essential Element of Information 9 in EPA 1997. They also contain information similar to that in OCRWM's Quality Assurance Requirements, Sections I.2.6B, "Configuration Control," I.2.6C, "Configuration Status," and I.2.7 "Defect Reporting and Resolution" in OCRWM 1995.)

4.2.1 Change Request

A change request may originate from a client, a project team member the client has contacted, or a project team member who has identified a need for a modification. The process serves several purposes. It

- provides information on the potential location of the problem or enhancement
- identifies the problem or enhancement
- provides information to determine the priority of the problem or enhancement
- documents under what circumstances the problem occurs
- provides for concurrence by the module/system framework manager and subject-matter expert.

Sometimes circumstances prevent the suggested change from being implemented. For example, enhancements may be suggested for which a client is uninterested in paying, or someone may report an error in an outdated version of code. However, the change request is documented regardless of whether the change is actually implemented. This documentation enables staff to track issues that, while not key to improvements today, may require action later.

A change request may involve more than one problem/enhancement; in some cases, several related problems are reported at one time. The request is duplicated later in the process when each change is assigned a method of correction.

Once a change request has been initiated, it is routed through the integration leader, who will assign a tracking number, enter the number into a database, and distribute information to the affected code custodian. Sometimes it is unclear as to which modules are involved/affected by the

proposed change. In this case, the integration leader assigns a temporary tracking number until the change can be further evaluated by the module/system framework manager.

4.2.2 Change Documentation

Once a change has been documented in the change request process, the code custodian completes the evaluation of the change and submits it to the subject-matter expert and module/system framework manager for approval and signature. (At this point, multiple change requests that were submitted together can be distributed so that each change can be tracked separately.) The change attachment serves to document

- the evaluation of the problem or enhancement by the subject-matter expert(s)
- changes made to the code
- tests run to ensure that changes work properly
- team members involved in the decision and change process.

If the change is denied, the module or system framework manager provides the completed change package to the integration leader for archiving. If the change is approved, the code custodian begins design and implementation of the proposed change. The module manager then requests a tester to test the changes made with the baseline set of scenarios and any other testing scenarios necessary to confirm the expected effects of the change. The code custodian also provides information to explain what changes were implemented and how. Examples of such documentation include print outs of screens, code comparisons, etc. The code custodian completes the change package and provides it to the integration leader for archiving, keeping a copy.

If a solution triggers another problem or needed enhancement, the module/system framework manager makes another change request and the process begins again.

4.2.3 Change Request Summary

When a change has been implemented, the code custodian keeps a copy of the completed change package and returns the original to the integration leader, who prepares a change request summary. This package serves to

- summarize the information from the change request form
- document reviews and approvals of subject-matter experts and module/system framework managers
- document source code and backup updates.

The integration leader files the change request, change package, change request summary, and any associated documentation for later reference. Copies of requests originated by a client are also placed in the client folder.

4.3 Design and Development

Modification design and development takes a similar approach to that discussed for new software development (Section 3.0). After evaluating the modification and its potential impacts to existing code and results, the software is modified and tested. The project team might also consider the following design issues:

- impacts to software user's guidance
- information for updating the software development package
- potential additional reviews.

5.0 Software Integration, Testing, and Evaluation

Software engineers test all environmental software that the project team develops before it is used outside Pacific Northwest. Each component is individually tested, and the entire system is tested to ensure compatibility. The tester develops the software test package (see Appendix C) based on the requirements for the system framework or module being tested. An independent tester may also evaluate the test package to ensure completeness and accurate results.

Three main objectives are associated with testing:

1. To uncover errors in requirements, design, and programming so that these errors can be correct before software implementation.
2. To identify or confirm limitations of the system so that these limitations can at a minimum be documented (perhaps in a revision to the design documentation) for future reference, and at best eliminated so as not to impair functionality of the software.
3. To build confidence in the capabilities of the software to meet its requirements.

Testing can often produce additional change to already baselined software. In this case, recommended changes to software are routed through the modification process to ensure the feasibility of the recommended changes (the modification process is discussed in Section 4.0). The testing and evaluation of results is handled slightly differently for new software versus changes to existing software, as detailed below.

5.1 New Software

Testing of newly developed software is key in the environmental software quality process (see Figure 3.1). Tests are documented in the software test package. Subject-matter experts and software engineers provide information to the tester to summarize the reason for performing the tests and contribute to the scope of the test. The foundation of testing is the requirements outlined in the requirements package and software development package. (The software test package and test results contain similar information to EPA's "Software Test and Acceptance Plan" and "System Integration Test Reports," Essential Elements of Information 7 and 12, respectively, in EPA 1997. They also contain information similar to that in OCRWM's Quality Requirements, Section I.2.2, "Software Verification and Validation," and I.2.4, "Software Validation" in OCRWM 1995.)

The software test package will contain

- a brief description of the system framework, module, or program being tested
- the scope of the testing to be conducted
- the relationship between specific test cases and the software requirements
- the major tasks, activities, techniques, and tools necessary to prepare for and perform testing
- characteristics and configurations of any hardware or software necessary to support tests
- test case specifications, including the unique identifier assigned to each case, the rationale for selecting each case, and all the input data and relationships between input items required to conduct each test
- the expected results for each test case and a statement of whether the results are acceptable.

In addition, the software test package may contain detailed procedures for conducting the test, if needed. Such procedures may describe

- the sequence of actions necessary to prepare for and conduct the test
- methods or formats for logging the results of test execution and test incidents
- how test measurements would be made
- the sequence of actions necessary for shutdown, restart, execution halt, and restoration of the hardware/software environment
- the actions necessary for addressing anomalies that may occur during execution of the test.

Software testing can be performed at both the unit and system levels. Unit testing evaluates individual components in isolation from other components. Unit testing is primarily employed at the subroutine and program levels and is first performed informally (i.e., not documented) by the programmer as part of the code-development process. Formal unit testing is then performed, in which a test plan is prepared and the testing results documented.

Unit testing is followed by system testing, in which the performance of groups of components functioning together is evaluated. System testing evaluates data communication between the components comprising the software (also called integration testing), as well as the overall performance of the software. Issues that arise during system testing are dealt with by fine-tuning how individual components run so that they work together properly as a team.

Success at any level of testing depends on the quality of testing performed at the previous level. For example, system testing of a module will not go smoothly if unit testing of the programs composing that module was not properly completed.

It is important to keep in mind that complete testing of any software is not possible. For software of even modest complexity, the set of all possible cases (i.e., all possible input data and combinations of input data items) is essentially unlimited. Therefore, the testing process can be thought of as a type of statistical analysis. The tester attempts to 1) select a representative sample of cases from the set of all possible cases, 2) evaluate the software's performance on the sample cases, and then 3) make an inference about the software's performance on the complete set. This last step involves uncertainty. Showing that software works properly on the sample cases does not mean it will work properly on all possible cases. However, successful testing does provide confidence that the software meets its functional requirements and will perform acceptably on most of the possible cases.

Quality testing also requires an in-depth knowledge of the software being tested. This knowledge includes an understanding of the input data, the solution technique implemented by the software, and what the output data should look like. Quality testing also requires that it be done independently from the software developers. Knowledge of the software developer's thought process should not be necessary to verify that the code works properly and logically.

If problems are discovered during testing, the process detailed in Section 4.0 does not need to be followed explicitly to resolve the problem, because newly developed software has not already been baselined. Instead, the tester notifies the lead software engineer of the problem. The lead software engineer rectifies the problem and provides the modified system framework, module, or code to the tester so that testing can resume. At the conclusion of the testing, the tester completes the software test package and has it reviewed by the subject-matter expert and lead software engineer. Once approved, the package documents the baseline test cases and their results, and the system framework or module is baselined.

5.2 Modified Software

Modifications to baselined system frameworks or modules should proceed according to the process outlined in Section 4.0. If the modification affects the expected results of any of the baseline test cases, the tester revises the software test package to state the new expected results. The tester then reruns the baseline test cases to confirm the expected effects of the modification. In addition, other test cases may be prepared and executed, as appropriate, to specifically test the modification. The subject-matter expert and module/system framework manager decide whether any additions, deletions, or revisions to the baseline test case set are warranted. The results of both new tests and the standard baseline test cases are detailed in the modification documentation (baseline test run section of the software change attachment shown in Appendix B).

5.3 General Test Scenarios

General test scenarios are prepared for new software and existing software that requires retesting. These scenarios are described as the minimum set of scenarios that is necessary to ensure the correctness of the software produced.

Scientific staff with software coding capabilities who have not worked on the development of a particular piece of software develop and implement the testing program. Such staff provide an independent, yet credible, verification that the software will perform as expected. Often specific staff follow a piece of software through development and initial testing into modifications and retesting.

For a system framework, the scenarios evaluated are based on user friendliness and module accessibility. Examples of areas tested for system frameworks include correct file execution, accurate file communication, and user understandability.

For a module user interface, the scenarios evaluated are based on the user friendliness and accuracy in data gathered for the model. Examples of areas tested for the module user interface include representation of model capabilities, accurate file communication, and user understandability.

For a model, the scenarios evaluated are based on the formulations and purpose of the model. Examples of areas tested for the models include correct implementation of formulations, accurate communication of input/output data, and potential combination of options.

For pre/post-processors, scenarios evaluated are based on correct data transfer and compliance with system framework file specifications.

6.0 Software Implementation

The last phase of ensuring quality in software development occurs in software implementation. However, the expectations for software implementation were first introduced in the requirements-analysis phase of the project. Implementation means that the specific software developed for that client can be transferred to the client to be applied at their organization by their own staff. How software was developed, along with user acceptance and operational constraints, as discussed below, influence how this software is implemented.

6.1 Technology Transfer (see Appendix C)

The transfer of usable software to the client is based on the details depicted in the statement of work or project management plan. Client agreements are often arranged with a requirement that the software be user friendly and operable by someone who is not part of the project team. The level of client implementation support is arranged to aid the end user of the software tool in adjusting to the look and feel of the software.

The subject-matter expert is responsible for determining whether the software is ready to be delivered to the client. This determination involves confirming the completeness and accuracy of the requirements package, the software development package, the software test package, and the user's guidance.

6.1.1 Client Implementation Support

Project agreements often include a client implementation support phase that comprises the delivery of software to a client and a trial time with client support through initial software usage. This support may be offered over the phone, at the client location, or in training sessions on the use and foundations of the software being delivered. Technical support can also be negotiated beyond the initial project agreement to enable a client to expand on software capabilities or to aid in the client's application of the software tools.

6.1.2 Implementation Documentation

Implementation documentation is usually offered as an online feature of the software with additional documentation of file specification and formulations also available. The software development "kit" includes information to help the user understand how to troubleshoot the software. Additional information on software user's guidance can be found in Section 3.0.

6.1.3 User Training

Training on the appropriate use of the software is recommended. This training should include information such as the capabilities of user interfaces, the reasoning for formulations that were implemented, and the limitations of the software produced. This training may occur through a formal training session, technical support of individual users, an online tutorial, or some other form desired by the client and agreed to by the project team.

6.2 Software Operations and Maintenance

Support to software operations and maintenance depends on the scope of the project. In some cases, the operation and maintenance is included in the project and therefore falls into the modification aspects of this document (Section 4.0). In other cases, the project ends upon delivery of software to the client; in this case, the client could negotiate continued software maintenance for future use. Guidance of the proper use and operation of the software is provided in the user training as discussed above. Documentation packages gathered for developed software are maintained and stored for the life of the product.

7.0 References

Office of Civilian Radioactive Waste Management (OCRWM). 1995. *Quality Assurance Requirements and Description, Software*. U.S. Department of Energy, Washington, D.C.

Standards Based Management System. 1997a (and as updated). "Computer Software and Database Control Standard." <http://sbms.pnl.gov:2080/standard/94/9400t010.htm>, Pacific Northwest National Laboratory, Richland, Washington.

Standards Based Management System. 1997b (and as updated). "Quality Assurance Planning Standard." <http://sbms.pnl.gov:2080/standard/87/8700T010.htm>, Pacific Northwest National Laboratory, Richland, Washington.

U.S. Environmental Protection Agency (EPA). 1997. *System Design and Development Guidance*. EPA Directive Number 2182, Washington, D.C.

Whelan, G., K. J. Castleton, J. W. Buck, G. M. Gelston, B. L. Hoopes, M. A. Pelton, D. L. Streng, and R. N. Kickert. 1997. *Concepts of a Framework for Risk Analysis in Multimedia Environmental Systems (FRAMES)*. PNNL-11748, Pacific Northwest National Laboratory, Richland, Washington.

Appendix A
Roles and Functions of Project Team Members

Appendix A—Roles and Functions of Project Team Members

A multidisciplinary project team develops, tests, and documents each new piece of software for risk analysis in multiple environmental media or major revision to an existing piece of software. Depending on software requirements and available resources, an individual might fill more than one role on a team, or serve different roles on different teams. In addition, team members may change over time. However, each team will have at least one subject-matter expert and integration leader, as well as either a lead software engineer and project manager for new software or a module/system framework manager and code custodian for modifications to existing software. Except for code custodians and module/system framework managers, the individual assigned for each role is determined by the project manager at the onset of a specific project. The project manager, working with line managers, chooses code custodians and module system framework managers once software has been baselined (i.e., designed, developed, and tested).

Specific roles and functions for various team members are outlined below in alphabetical order. At the end of this list, Table A.1 shows key activities and the project team member responsible.

Code Custodian

- **Definition:** team member responsible for maintaining baselined software under direction of the module/system framework manager
- **Modified Software Design and Development Phase Responsibilities:** complete change requests; attempt to reproduce problems; determine potential ways to solve problems; evaluate change for effect on other segments of the code; implement or lead implementation of the change; complete change package; keep a copy
- **Testing Phase Responsibilities:** help determine whether additional tests are needed; revise code based on test results as needed
- **Implementation Phase Responsibilities:** assist in revising user's guidance and training as needed; serve as reference for integration leader in supporting implementation; keep a copy of source code file locations

Integration Leader

- **Definition:** team member responsible for ensuring appropriate quality throughout the software life cycle as well as project management
- **Requirements-Analysis Phase Responsibilities:** assist in developing project management plan or statement of work; assist in developing requirements package; review requirements package
- **New Software Design and Development Phase Responsibilities:** assist in developing software development package
- **Modified Software Design and Development Phase Responsibilities:** assign tracking number to change request, enter information into database, and distribute information to affected code custodian; ensure changes that require additional evaluation are resolved; ensure completeness of change package; complete change request summary; alert users as needed to changes
- **Testing Phase Responsibilities:** assist in developing and implementing test process as well as software test package
- **Implementation Phase Responsibilities:** develop user's guidance and training as needed to support client implementation; serve as primary point of contact for general user support; keep all documentation files

Lead Software Engineer

- **Definition:** team member responsible for developing a system framework or module (including user interface, models, pre/post-processors as needed, and file transfer specifications and related software); ensure that all components are understandable and compatible, and provide the correct inputs and outputs
- **Requirements-Analysis Phase Responsibilities:** none, unless input is requested by the subject-matter expert
- **New Software Design Phase Responsibilities:** lead team of other software engineers (or conduct design alone for less complex pieces of software) in designing software to meet requirements; determine appropriate databases and file structures for input and output; develop additional specification documentation as needed; confirm relationship between requirements and design elements with subject-matter expert; complete design portion of software development package
- **New Software Development Phase Responsibilities:** lead team of software engineers (or develop software alone for less complex pieces) in programming software; complete the software development package
- **Testing Phase Responsibilities:** provide information to the tester to aid in summarizing the reason for the test as well as the scope of the test; respond to test results; review software test package
- **Implementation Phase Responsibilities:** assist in developing user's guidance and training for the software as needed; serve as reference for integration leader in supporting client implementation

Module/System Framework Manager

- **Definition:** team member responsible for maintaining a specific module or system framework, including user interface, underlying scientific models, and pre/post-processors.
- **Requirements-Analysis Phase Responsibilities:** understand client and user requirements for module
- **Modified Software Design and Development Phase Responsibilities:** initiate change request; review potential ways to solve the problem; approve changes with the subject-matter expert; provide change document to integration leader should subject-matter expert deny changes; review implemented change; review change package for module; assist development of change request summary
- **Testing Phase Responsibilities:** determine with subject-matter expert whether additional test cases are necessary; review reruns of test cases
- **Implementation Phase Responsibilities:** assist in developing user's guidance and training as needed to support client implementation

Project Manager

- **Definition:** team member responsible for assembling project team, interfacing with the client, communicating client needs to the rest of the project team, ensuring that all Laboratory and client requirements are met throughout the project, and providing leadership throughout the project
- **Requirements-Analysis Phase Responsibilities:** work with client and subject-matter expert to identify requirements and needs; develop project management plan or statement of work and gain internal as well as needed external approvals; communicate requirements to team members; review requirements package; ensure project will meet Laboratory and client needs
- **Software Design Phase Responsibilities:** review design portion of the software development package before implementation

- Software Development Phase Responsibilities: ensure project is meeting Laboratory and client needs; review completed software development package; work with line managers to identify module/system framework manager and code custodian for baselined code
- Testing Phase Responsibilities: ensure that project is meeting Laboratory and client needs
- Implementation Phase Responsibilities: ensure that project is meeting Laboratory and client needs; ensure appropriate closeout of project activities

Software Engineer

- Definition: team member responsible for developing a particular section of code to support a specific module or the system framework
- Requirements-Analysis Phase Responsibilities: none
- New Software Design and Development Phase Responsibilities: design and develop code under the leadership of the lead software engineer; use object-oriented programming techniques
- Modified Software Design and Development Phase Responsibilities: assist in initiating a change request as needed; design and develop code under the leadership of module/system framework manager; use object-oriented programming techniques
- Testing Phase Responsibilities: assist in developing software test package as needed; revise code as needed based on test results
- Implementation Phase Responsibilities: serve as reference for integration leader in supporting client implementation

Subject-Matter Expert

- System Framework Definition: team member responsible for ensuring the appropriateness of the framework in addressing client and user needs; ensuring that framework specifications are accurate and documented; ensuring that the user interface is logical and predictable; and providing technical leadership
- Module Definition: team member responsible for ensuring that the underlying scientific models for a particular module are scientifically accurate and predictable and provide useful results; and for providing technical leadership
- Requirements-Analysis Phase Responsibilities: assist in developing project management plan or statement of work; determine requirements for framework or module; develop requirements package for framework or module
- System Framework Design and Development Phase Responsibilities: help software engineer to understand requirements, their relationship to design elements, and their relative importance; approve design for new software; review completed software development package; review and approve baselined software
- Module Design and Development Phase Responsibilities: provide formulations and model information for new software; approve design for module for new software; review and approve modifications to baselined software
- New Software Testing Phase Responsibilities: collaborate with the software engineer to provide information to help the tester summarize the reason for the test as well as the scope of the test; review software test package to ensure that model results are appropriate
- Modified Software Testing Phase Responsibilities: decide with manager changes to test cases; review software test package to ensure model results are appropriate
- Implementation Phase Responsibilities: assist in developing user's guidance and training as needed; determine when software is ready for client delivery

Task Leader

- Definition: team member responsible for ensuring that all activities associated with a particular task are accomplished on time, within budget, and according to client and Laboratory requirements
- Requirements-Analysis Phase Responsibilities: assist in developing project management plan or statement of work; understand client and user requirements for task; assist in developing requirements package
- Software Design and Development Phase Responsibilities: review design portion of software development package; review completed software development package; ensure that task contributes to overall project
- Testing Phase Responsibilities: ensure that task contributes to overall project
- Implementation Phase Responsibilities: ensure that task contributes to overall project; ensure appropriate closeout of task activities

Technical Reviewer

- Definition: team member responsible for providing a “second opinion” on the scientific accuracy of a specific model, its predictability, and/or its ability to provide useful results or to provide a similar opinion on the utility of coding in a system framework, module user interface, model, or pre/post-processors.
- Requirements-Analysis Phase Responsibilities: understand client and user requirements; review requirements package if needed
- Software Design and Development Phase Responsibilities: review software development package if needed for new software; review change package if needed for modified software
- Testing Phase Responsibilities: review test results if needed
- Implementation Phase Responsibilities: review user’s guidance and training as needed

Tester

- Definition: team member responsible for ensuring that software meets requirements and for documenting results of tests
- Requirements-Analysis Phase Responsibilities: understand client and user requirements; review requirements package
- New Software Design and Development Phase Responsibilities: review design portion of the software development package; understand design and development constraints
- Modified Software Design and Development Phase Responsibilities: as directed by the module/system framework manager, test changes by rerunning baseline test cases as well as developing and running new tests as appropriate
- Testing Phase Responsibilities: develop software test package; implement testing process and provide documentation of test results for both new and modified software; notify lead software engineer of problems
- Implementation Phase Responsibilities: serve as reference to integration leader to support client implementation

Table A.1. Key Activities During the Software Life Cycle and Team Members Responsible*
 (Note: Shading in cells highlights key responsibilities.)

| Project Team Member | PMP or SOW | Requirements Package | Software Development Package | Software Test Package | User's Guidance or Training | Change Request | Change Package | Change Request Summary |
|---------------------------------|------------|----------------------|------------------------------|-----------------------|-----------------------------|----------------|----------------|------------------------|
| Code Custodian | | | | | | C | R | A |
| Integration Leader | A | A, r | A | A | R | A | | R |
| Lead Software Engineer | | | R | A, r | A | | | |
| Module/System Framework Manager | | | | r, for revision | | I | r | A |
| Project Manager | R | r | r | | | | | |
| Software Engineer | | | A | A | | A | A | |
| Subject-Matter Expert | A | R | approve design | r | A | approve | | A |
| Task Leader | A | A | r | | | | | |
| Technical Reviewer | | r | r | | r | | r | |
| Tester | | r | r | R | | | A | |

*Abbreviations used in table: in heading row—PMP = Project Management Plan; SOW = Statement of Work. In other rows: C = completes; R = responsible; A = assists responsible team member; r = reviews as needed; I = initiates.

Appendix B
Example Software Development and Modification Forms

Software Development Package

1. OVERVIEW

SOFTWARE NAME: _____ SOFTWARE # _____

{1.1} Type of code: ___ New ___ Replacement ___ Upgrade

{1.2} Component Development Team:

Task Manager _____ Integration Leader _____

Subject-Matter Expert _____ Lead Software Engineer _____

Tester _____ Technical Reviewer _____

Software Engineers _____

+----- {1.3} Description of Code Scope -----+

+-----

{1.4} Deliverables

Due Date

| | |
|-----------------------------------|-------|
| ___ Requirements Package | _____ |
| ___ Software Development Package | _____ |
| ___ Baseline Software Development | _____ |
| ___ User's Guidance | _____ |
| ___ Software Test Package | _____ |
| ___ Other: _____ | _____ |
| ___ FINAL REVIEW | _____ |

Project Manager Approval: _____ Date _____

Software Development Package (contd)

2. CODE REQUIREMENTS

___ {2.1} Requirements package attached.

___ {2.2} Draft mathematical formulations have been reviewed internally by the following individuals. Attach any formal comments and note sections requiring additional review.

| Print Name | Signature | Date | Sections Approved | Sections Needing Ad. Review |
|------------|-----------|-------|-------------------|-----------------------------|
| _____ | _____ | _____ | _____ | _____ |
| _____ | _____ | _____ | _____ | _____ |

___ {2.3} Draft mathematical formulations have been reviewed externally by the following individuals. Attach any formal comments and note sections requiring additional review.

| Print Name | Signature | Date | Sections Approved | Sections Needing Ad. Review |
|------------|-----------|-------|-------------------|-----------------------------|
| _____ | _____ | _____ | _____ | _____ |
| _____ | _____ | _____ | _____ | _____ |

Subject-Matter Expert: _____ Date: _____

3. CODE DESIGN AND DEVELOPMENT

___ {3.1} Baseline hard copy of source code listing attached.

___ {3.2} Diskette (labeled with activity number, component name, version, component developer name, and date) enclosed.

- ___ {3.2a} Source Code
- ___ {3.2b} Executable
- ___ {3.2c} "Readme" files with any special instructions

___ {3.3} Computer programming language used: _____
Additional languages used: _____

Code Custodian: _____ Date: _____

4. CODE TESTING

___ {4.1} Software Test Package attached.

Tester: _____ Date: _____

Software Development Package (contd)

5. USER'S GUIDANCE

___ {5.1} Software User's Guidance attached.

___ {5.2} User Documentation has been reviewed by the following. Attach any formal comments.

| Print Name | Signature | Date | Sections Approved | Sections Needing Ad. Review |
|------------|-----------|-------|-------------------|-----------------------------|
| _____ | _____ | _____ | _____ | _____ |

Integration Leader: _____ Date: _____

6. BACKUPS

___ {6.1} Backup made to disk and stored in another location.

Location: _____

Date: _____

___ {6.2} Backup made to a network fixed disk located in separate building.

Location: _____

Date: _____

___ {6.3} Hard copy of source code stored in another location.

Location: _____

Date: _____

Code Custodian: _____ Date: _____

Software Change Request

Project _____

Version _____

Software (Indicate)

System Framework: _____

Module: _____

Other: _____

Unknown: _____

Change Submitted By: _____

Date: _____

Organization: _____

Phone: _____

Requested Completion Date: _____

Fax: _____

Nature of Request:

Problem

Enhancement

Error

Mistake

Change

Enhancement(s) or Problem(s) Reported

Priority:

Critical

Important

Routine

List of Attached Problem Documentation

___ Input file that demonstrates problem

___ Error message

___ Output file annotated to demonstrate problem

___ Other _____

Route To:

Date: _____ Integration Leader: _____

Date: _____ Subject-Matter Expert: _____

Date: _____ Code Custodian: _____

Date: _____ Tester: _____

Change Attachment

Project _____

Version _____

Software (Indicate)

System Framework: _____

Module: _____

Other: _____

Unknown: _____

EVALUATION OF PROBLEM/ENHANCEMENT

Date Received: _____

Evaluated by: _____

Date: _____

Approved Disapproved

Subject-Matter Expert

Date: _____

CHANGES MADE

Date Received: _____

Code Custodian: _____

Date: _____

BASELINE TEST RUN

Date Received: _____

Tester _____

Date: _____

_____ Version before change

_____ Version after change

UPDATE OF USER INTERFACE REQUIRED

Change Request Summary

Project _____ Version _____

Software (Indicate) System Framework: _____
Module: _____
Other: _____
Unknown: _____

Change Requested/Submitted by: _____

Date Submitted: _____ Requested Completion Date: _____

Nature of Request: _____ Problem _____ Enhancement

Summary of Enhancement(s) and/or Problem(s)

Attached Documentation

- ___ Change Request
- ___ Input file that demonstrates problem
- ___ Error message
- ___ Output file annotated to demonstrate problem
- ___ Software Change Attachment
- ___ Output file annotated to demonstrate correction
- ___ Listing before changes
- ___ Listing after changes
- ___ Sample problem output

Notification of Subject-Matter Expert, Code Custodian, and Module/System Framework Manager

Disks Updated

- ___ Version after change
- ___ Version after change
- ___ Distribution package
- ___ Source Code Configuration Control
- ___ Master file list updated

Creation Date of .EXE Files:

Integration Leader: _____ Date: _____

Checklist for Quality Assurance Documentation

General Requirements Analysis

--Documented in (choose one or more of the following and note location of document)

- Project Management Plan
- Task Plan
- Statement of Work
- Change Request Form (for modified software)

--Contains information on (all of the following)

- problem description
- deliverables
- project team
- capabilities to be used
- restrictions
- difficulties envisioned
- compatibilities with existing software/hardware

Specific Requirements Analysis

--Documented in (choose one or more of the following and note location and/or PNNL number of document)

- outline
- flowchart
- Work Breakdown Structure
- requirements document
- presentation materials

--Contains information on (all of the following)

- purpose of the software
- structure of the software
- hardware and software requirements
- input and output requirements
- scientific basis
- assumptions
- limitations

Design Documentation

--Documented in (choose one of the following and note location and/or PNNL number of the document)

- Design portion of Software Development Package
- Design Document
- Change Attachment Form for modifications to existing software

--Contains information on (all of the following)

- code type and description
- development team members
- specifications
- logic diagrams
- "help" descriptions
- methods to ensure consistency in components
- mathematical formulations
- need for pre/post-processors

Development Documentation

--Documented in (choose one of the following and note location and/or PNNL number of document)

- Programming portion of Software Development Package
- Specifications Document
- Change Request Summary for modifications to existing software

--Contains information on (all of the following)

- baseline hard copy of the source code
- diskette copy
- name of computer language(s) used

Testing Documentation

--Documented in test package, which may include one or more of the following (note location and/or PNNL number of document)

- test plan
- example results of testing
- detailed results of testing
- procedures for testing
- Contains information on (all of the following)
 - description of software
 - testing scope
 - relationship between test cases and requirements
 - test activity description
 - hardware and software needed to implement plan
 - test case specifications
 - expected results

User's Guidance

--Documented in (choose one or more of the following and note location and/or PNNL number of document)

- online help
- hardcopy printout
- tutorial
- training
- Contains information on (all of the following)
 - description of software
 - description of use of user interface
 - mathematical formulations
 - example problems
 - explanation of modules included

General Quality Assurance Documentation

--Documented in (choose one or more of the following and note location or PNNL number of document)

- Quality Assurance Program Document
- Quality Assurance Software-Specific Checklist
- Quality Assurance Project-Specific Guidance
- Project Management Plan
- Contains information on (all of the following)
 - purpose of quality assurance program
 - client-specified activities
 - activities required to ensure quality in software

Quality Assurance Archive

--Documented in

- hard-copy files (note location and custodian)
- backup files in multiple storage locations (note locations and custodian)
- Contains information on (all of the following)
 - all quality assurance documentation
 - client correspondence regarding software
 - modifications made to baselined software
 - disk copy backup

Completed by _____ Date _____

Approved by
System/Module Manager _____ Date _____

Appendix C
Glossary

Appendix C—Glossary

The following terms, which are listed in alphabetical order, are used in this document in ways that may not be familiar to some readers.

baseline—software that has been designed, developed, and initially tested to meet a particular need, either internal to the Laboratory or for a client

black box testing—the tester does not know how the process occurs (all he/she can test is that good data go in and data one might expect comes out)

change request—a written request (or a verbal request written down) by a client or other user of environmental software to modify that software. Changes can include enhancements or revisions to solve identified problems. Each change is tracked separately.

change request summary—summary of all activities related to a particular change in a module or system. The summary is completed by the integration leader.

code—information provided to guide a computer system through a particular process. Both modules and system frameworks depend on code to function properly.

component—a piece of software, for example, module, module user interface, model, pre/post-processor, or framework user interface.

design—the process of determining appropriate ways of meeting client and user requirements

development—the process of implementing a design

environmental software—a set of modules or a framework designed to analyze risk to human health or the environment through multiple environmental media

error/bug—a difficulty encountered by the user when applying software that hinders the application for which the software was intended

framework—mechanism for holistically linking a variety of models while ensuring compatible input and outputs and maintaining the development of testing legacy of individual components

hard wired—software that has been developed with narrowly defined coding such that modifying, enhancing, or otherwise adapting the code for better use is extremely difficult and compromises the functionality and credibility of the software

input—data, information, or code that is needed by the software to function

implementation—the act of transferring software to a client for application at their organization

model—the set of scientific calculations that defines a particular module

modification—a change to or the process of changing baselined software. Modifications can be enhancements or revisions to fix a particular problem such as an error or bug.

module—a user interface, scientific model, and possibly pre/post-processors that together provide a tool for analyzing environmental phenomena; these modules may be related to a particular environmental media (water, air, soil, etc.), be health-risk-based, be statistically based, or contain any other set of scientific formulations used to model the environment.

module user interface—the computer screens that make it easy to collect the data necessary to run the model. Besides gathering the necessary data, the user interface often provides online help to the user, lists storage options for collected data, provides flexible unit inputs, and allows access to other user support functions.

multiple media—potential portions of the environment through which a human, animal, or plant might be exposed to contamination (for example, water, air, soil, etc.)

object-oriented—the practice of breaking down software into its unique and logical components, then developing those components only once and applying them to the software. Each individual component's purpose is well defined and testable.

output—data, information, or code that results from a module functioning

package—the information, documentation, print outs, diagrams, and/or electronic files that document the activities conducted under a particular project or task for a particular stage in the software development life cycle (requirements package for requirements analysis, software development package for design and programming, software test package for testing, user's guidance and training package for implementation, and change package for modifying baselined software).

pre/post-processor—additional code necessary to reconcile module input/output with output/input needs from the system framework.

project management plan—description of the ways in which an activity will be conducted to meet client needs

project team—various staff assigned to develop software to meet client needs

requirements—objectives, activities, constraints, and other information that specify what the client needs the software to accomplish

requirements package—the information, documentation, printouts, diagrams, and/or electronic files that document the objectives, activities, constraints, and other information that specify what the client needs the new system or module to accomplish and how the client will use the software

software development package—the information, documentation, printouts, diagrams, and/or electronic files that document the methods by which client requirements will be met

software test package—the information, documentation, printouts, diagrams, and/or electronic files that document the how the system, module, or change will be tested to ensure that it meets client requirements

statement of work—description of activities to be accomplished for a particular client and the methods by which they will be accomplished

technology transfer—the process of providing software to the client's organization for use by their staff

testing—the process of checking that systems or modules meet the client requirements and function within the necessary settings

user—the person who will use the software to estimate risks to human health or the environment

white box testing—the testor follows the data through all the transformations and checks that the code came up with

Appendix D
Outlines of Example Documents

Appendix D--Outlines of Example Documents

The following are outlines of documents described in the text that might be used to describe requirements, design elements, specifications, and testing for software, based on client needs.

Requirements Document Outline

1.0 Introduction

- Describe purpose of project
 - What problem is the software going to solve?
 - How will the software be applied?

- Describe purpose of document

2.0 System-Level Requirements

- Describe the purpose of the software
- Describe the structure of the software
 - Major subsystems
 - Modules with models and pre/post-processors
 - Flow chart of the software
- Summarize system-level requirements
- List hardware and software requirements, including memory, processor speed, system platform, etc.

3.0 to n.0 Component Requirements

- Describe the purpose of each component
- Summarize the requirements of each component
- List input requirements
- List output requirements
- List scientific requirements
 - List mathematical formulations that must be used
 - List assumptions on which the science is based
 - List limitations of the component

References, if used

Appendixes as needed

- Glossary and definition of acronyms and abbreviations

Design Document Outline

1.0 Introduction

- Describe purpose of project
 - What problem is the software going to solve?
 - How will the software be applied?

- Describe purpose of document

2.0 Summary of System-Level Requirements

- Describe the purpose of the software
- Describe the structure of the software
 - Major subsystems
 - Modules with models and pre/post-processors
 - Flow chart of the software
- Summarize system-level requirements

3.0 to n.0 Description of Components

Summarize requirements for each component (system framework, module, model, pre/post-processor, etc.)

Describe structure of component

Provide a picture (flowchart or diagram) of the component

Describe the user interface

Describe how the component will be implemented (basic flow of information through it)

Describe database files for the component

How they will be populated

How they will be coded

How they will be structured

Input/output specifications

References, if used

Appendixes as needed

Glossary and definition of acronyms and abbreviations

Specifications Document Outline

1.0 Introduction

Describe purpose of project

What problem is the software going to solve?

How will the software be applied?

Describe purpose of document

2.0 System-Level Requirements

Describe the purpose of the software

Describe the structure of the software

Major subsystems

Modules with models and pre/post-processors

Flow chart of the software

Summarize system-level requirements

3.0 to n.0 Component Specifications

Summarize requirements for each component (system framework, module, model, pre/post-processor, etc.)

Describe structure of component

Provide a picture (flowchart or diagram) of the component

List detailed specifications

Names of files

Coding conventions

Input/output specifications

References, if used

Appendixes as needed

Glossary and definition of acronyms and abbreviations

Testing Plan Outline

1.0 Introduction

Describe purpose of project

What problem is the software going to solve?

How will the software be applied?

Describe purpose of document

Describe the purpose of the software

Describe the structure of the software

Major subsystems

Modules with models and pre/post-processors

Flow chart of the software

Describe component to be tested and the codes which comprise the component (i.e., core program or model, module user interface, and pre-/post-processors).

Describe the depth of testing (i.e., white box, black box, or both [see Appendix C]) and the reason for testing to that level.

2.0 Requirements

List and number each requirement for the component.

Provide a table with the requirement number on one axis and the test case name on the other. In the body, check off which requirement is tested by which test case(s). This table should demonstrate that each requirement is tested by at least one test case.

| | | Test Case | | | |
|--------------|---|-----------|---|---|---|
| | | 1 | 2 | 3 | 4 |
| Requirements | 1 | ✓ | ✓ | | |
| | 2 | | ✓ | | |
| | 3 | | | ✓ | ✓ |
| | 4 | | ✓ | | |
| | 5 | | | ✓ | ✓ |
| | 6 | | | | ✓ |

3.0 Test Cases

Describe each test case and the rationale for selecting this case

List all the input data needed to run the case

Be specific by listing values and their units for all input variables

Be sure to use the same units that the component being tested requires

Describe the expected results of the case to provide acceptance criteria for deciding if the test was successful, the more specific, the better.

List any special procedures necessary to execute the test (i.e., deviations from the general procedure explained below in Appendix C).

Specify codes

Specify compilers

Specify linking protocols.

4.0 References

Appendix A - Expected Results Documentation

Provide hand or spreadsheet computations of expected results for each test case

Appendix B - Input Datasets

List the common input datasets, using the same units that the component being tested requires

Appendix C - General Procedure for Test Case Implementation

Describe how someone would set up and run a test case for this software

Provide file-format specifications if the tester is expected to populate input files by hand

Distribution List

No. of
Copies

No. of
Copies

OFFSITE

ONSITE

2 DOE Office of Scientific and
Technical Information

C. B. Nelson
U.S. Environmental Protection
Agency
J.S. Building, Rm 3102S
Washington, D.C.

G. F. Laniak
U.S. Environmental Protection
Agency
Environmental Restoration Lab
College Station Rd.
Athens, GA 30613

P. M. Beam
U.S. Department of Energy
Cloverleaf Building, Rm 2151
Germantown, MD 20874-1290

37 Pacific Northwest National
Laboratory

J.W. Buck K6-80
K.J. Castleton K6-80
W.C. Cosby K7-62
G.M. Gelston (20) K6-80
B.L. Hoopes K6-80
R.E. Lundgren K9-69
B.A. Napier K3-54
J.P. McDonald K6-96
K.R. Middleton K7-28
M.A. Pelton K6-80
D.L. Strenge K3-54
R.Y. Taira K9-33
G. Whelan K9-36
Information Release (5)

M98054110



Report Number (14) PNNL-11880

Publ. Date (11) 199805

Sponsor Code (18) DOE/EH; EPA, XF

UC Category (19) UC-630; UC-000, DOE/ER

19980707 090

DTIC QUALITY INSPECTED 1

DOE