DOCUMENTATION FOR THE FRAMES-HWIR TECHNOLOGY SOFTWARE SYSTEM, VOLUME 9: SOFTWARE DEVELOPMENT AND TESTING STRATEGIES

Project Officer
and Technical Direction:

Mr. Gerard F. Laniak
U.S. Environmental Protection Agency
Office of Research and Development
National Environmental Research Laboratory
Athens, Georgia 30605

Prepared by:

Pacific Northwest National Laboratory
Battelle Boulevard, P.O. Box 999
Richland, Washington 99352
Under EPA Reference Number DW89937333-01-0

U.S. Environmental Protection Agency
Office of Research and Development
Athens, Georgia 30605

October 1999

DISCLAIMER

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

This document was printed on recycled paper.
(9/97)

ACKNOWLEDGMENTS

A number of individuals have been involved with this effort. Mr. Gerard F. Laniak of the U.S. Environmental Protection Agency (EPA), Office of Research and Development, National Environmental Research Laboratory, Athens, Georgia, provided the overall technical direction and review throughout this work. This report was prepared by the Pacific Northwest National Laboratory[1] (PNNL) staff of John McDonald, Regina Lundgren, Randal Taira, Gariann Gelston, Gene Whelan, John Buck, Karl Castleton, Bonnie Hoopes, and Mitch Pelton. Additional PNNL staff supporting this effort include Wayne Cosby, Nancy Foote, Kristin Manke, Jill Pospical, Debbie Schulz, and Barbara Wilson. Useful inputs were provided by many U.S. EPA individuals working on the Hazardous Waste Identification Rule, including Messrs. Barnes Johnson, Stephen Kroner, and David Cozzie, and Drs. David Brown, Robert Ambrose, Zubair Saleem, Donna Schwede, and Sharon LeDuc, among many others.

(1) Operated by Battelle for the U.S. Department of Energy under Contract DE-AC06-76RLO 1830.

Summary

The U.S. Environmental Protection Agency (EPA) is developing a comprehensive environmental exposure and risk analysis software system for agency-wide application. The software system is being prototyped (i.e., initial design and implementation) for application to the technical assessment of exposures and risks relevant to the Hazardous Waste Identification Rule (HWIR). The software system being adapted to automate this assessment is the Framework for Risk Analysis in Multimedia Environmental Systems (FRAMES). The FRAMES-HWIR Technology Software System comprises a number of components, including the overall system software, five primary processors (one of which contains modules), and several databases. To ensure that all these components of the FRAMES-HWIR Technology Software System interact appropriately when placed into the software system, module and processor developers should meet a set of expectations in the areas of software development, quality assurance, and testing.

In the area of software development, module and processor developers are expected to

C design modules and processors to reformat and reorder data to meet the specific needs of the component

C provide a batch file as a single entry point to execute a module if pre- or post-processors are created for the module

C design modules and processors (as appropriate) to read the respective Site Simulation Files and Global Results Files Data Groups using the subroutines in the input/output dynamic link library (HWIRIO.DLL) provided by the Pacific Northwest National Laboratory (PNNL)

C design modules to read the appropriate file as needed from the Meteorological Database, using methods designed and implemented by EPA

C design modules to read the appropriate file as needed from the Chemical Database, using methods designed and implemented by PNNL

C design modules to write to the respective Global Results Files Data Group using the subroutines in the HWIRIO.DLL provided by PNNL

C design modules and processors to delete any temporary files before exiting such that the remaining files match the appropriate storage level

C design modules and processors to create an error file in flat-ASCII (American Standard Code for Information Interchange) format at the start of execution and delete it at the end of execution (provided that there are no errors produced) using the shared routines in the HWIRIO.DLL. If an error is produced for a module, then the module should report the error to the Multimedia Multipathway Simulation Processor (MMSP) module execution manager using the HWIRIO.DLL, which will then close the error file, and the module execution manager will terminate execution of the module and the MMSP. If an error is produced by a processor, then the processor should report the error to the System User Interface (SUI) using the HWIRIO.DLL, which will then close the error file and allow the SUI to terminate execution of the processor.

- C design modules and processors to create a warning file in flat-ASCII format at the start of execution and delete it at the end of execution (provided that there are no warnings produced) using the HWIRIO.DLL. If a warning is produced, then the module or processor should write a warning message to the warning file using the HWIRIO.DLL. A warning message will not cause the component execution to terminate.
- C design modules and processors to consume less than 16 Mb of RAM and 250 Mb of disk space, and have an execution time on the order of 1 second on a stand-alone PC (assuming a Pentium [586] 200 MHZ PC-based machine)
- C design modules to store a volume of data appropriate to an assigned data storage level (passed via a call argument)
- C use only C++ and FORTRAN 77 (or later version) programming languages for all models and pre- and post-processors accessing the HWIRIO.DLL. If any language other than these is used, it is the developer's responsibility to produce equivalent subroutines for file interfacing.
- C ensure that the module meets the specifications outlined in the document entitled *Documentation for the FRAMES-HWIR Technology Software System, Volume 8: Specifications* (Castleton et al. 1998; see the Bibliography section). To accomplish this, it may be necessary to include a pre- or post-processor as part of the module.
- C design, implement, and test pre- or post-processors if such are developed for a module.

In the area of quality assurance, module and processor developers are expected to

- C use an appropriate approach to quality assurance and documentation
- C work with related-media modelers to ensure consistency of assumptions/data transfer between media
- C provide documentation of the module and processor including user's guidance, mathematical formulations, and documentation of requirements, design/specifications, and testing.

In the area of testing, module and processor developers are expected to

- C develop a test plan according to the outline provided in Appendix A
- C ensure that unit testing of programs is thorough and well documented
- C ensure that integration testing of the pieces composing their processor is thorough and well documented
- C ensure that processors communicate with the wider system through the shared routines, such as the HWIRIO.DLL and the spatial and temporal integrator DLL and document such system tests
- C revise requirements documentation, design/specifications documentation, and program code implementation as needed to resolve issues found during testing
- C document module and processor limitations, addressing those limitations as needed in coding to ensure that the program functions as intended and eliminating those that inhibit the required functionality of the software

C provide documentation from internal testers to support the conclusion that the software meets its requirements

C follow the six-step process outlined in this document for internal testing

C provide the module or processor executable, the source code, the test plan, and results to an external independent test group for verification

C work iteratively with the external independent test group as needed to resolve issues.

Acronyms and Abbreviations

ASCII	American Standard Code for Information Interchange
EPA	U.S. Environmental Protection Agency
FRAMES	Framework for Risk Analysis in Multimedia Environmental Systems
HWIR	Hazardous Waste Identification Rule
HWIRIO.DLL	input/output dynamic link library
MMSP	Multimedia Multipathway Simulation Processor
PNNL	Pacific Northwest National Laboratory
SUI	System User Interface

Contents

Figures

1.0 Introduction

The U.S. Environmental Protection Agency (EPA) is developing a comprehensive environmental exposure and risk analysis software system for agency-wide application. The software system is being prototyped (i.e., initial design and implementation) for application to the technical assessment of exposures and risks relevant to the Hazardous Waste Identification Rule (HWIR). The HWIR is designed to determine quantitative criteria for allowing a specific class of industrial waste streams to no longer require disposal as a hazardous waste (i.e., "exit" Subtitle C) and allow disposal in Industrial Subtitle D facilities. Hazardous waste constituents with values less than these exit criteria levels would be reclassified as nonhazardous wastes under the Resource Conservation and Recovery Act.

The software system being adapted to automate this assessment is the Framework for Risk Analysis in Multimedia Environmental Systems (FRAMES), developed by the Pacific Northwest National Laboratory (PNNL). The FRAMES-HWIR Technology Software System is a nested software system that facilitates the simulation of site-based exposure and risk. The technology comprises a number of components, including the overall system software, five primary processors, and several databases (Figure 1.1). One of the processors, the Multimedia Multipathway Simulation Processor (MMSP; Figure 1.2), contains a module execution manager and a number of exposure and risk simulation modules that interact to develop a complete picture of contaminant movement through the environment to human and ecological receptors.

As can be seen in Figure 1.3, there are five levels of system organization. The highest level is the system, which encompasses the entire FRAMES-HWIR Technology Software System project. The system is defined as a group of processors controlled by an execution manager, in this case the System User Interface (SUI). A processor represents the second level and is defined as a group of modules that can be executed in many sequences, as determined by the module execution manager. A processor is characterized by a set of modules that exchange data through file specifications. A module represents the third level and is defined as a core program (i.e., model) and pre- or post-processors as necessary. These programs should always execute in sequence (i.e., pre-processor, program, post-processor). A program represents the fourth level. It consists of subroutines and is defined as a computer procedure for solving a problem, including collecting data, processing, and presenting results. Examples of programs discussed in this report are model, pre-processor, and post-processor. A subroutine represents the fifth level and is defined as a subprogram or a modularized portion of a program. The concept behind subprograms is that a complex problem can be made easier by breaking it down into smaller, less complex problems. Subroutines perform tasks such as reading and writing data and solving algorithms.

To ensure that all these components of the FRAMES-HWIR Technology Software System interact appropriately when placed into the software system, module and processor developers should meet a set of expectations in the areas of software development, quality assurance, and testing. This document summarizes for module and processor developers those expectations in the context of the FRAMES-HWIR Technology Software System. The following sections discuss guidelines for software development, quality assurance and control practices, and testing (including testing by software developers and independent testing). Companion documents for the system are listed in the references/bibliography section. Appendix A provides an outline of a test plan.

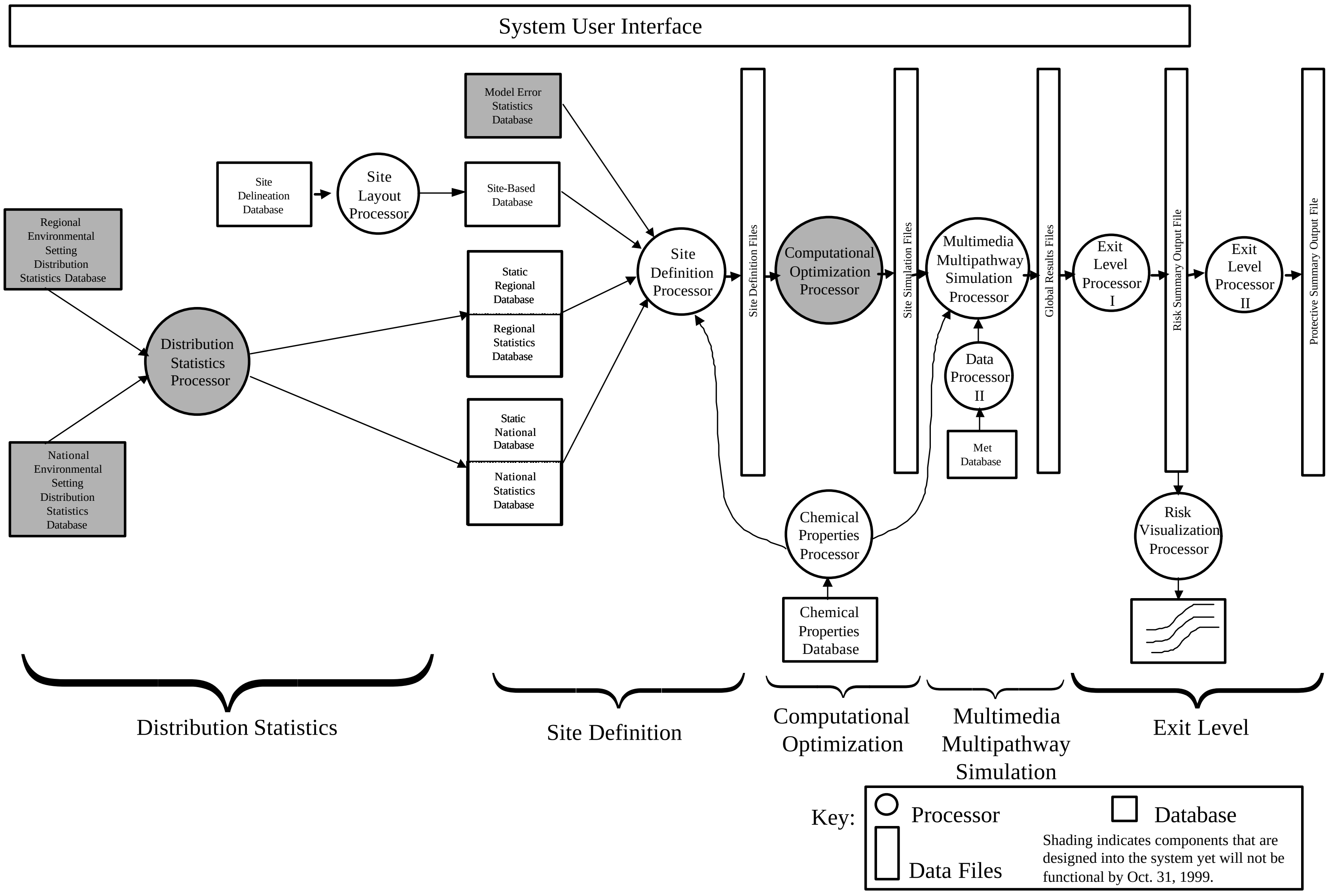

System User Interface
Model Error Statistics Database
Site Delineation Database
Site Layout Processor
Site-Based Database
Regional Environmental Setting Distribution Statistics Database
Distribution Statistics Processor
National Environmental Setting Distribution Statistics Database
Static Regional Database
Regional Statistics Database
Static National Database
National Statistics Database
Site Definition Processor
Site Definition Files
Computational Optimization Processor
Site Simulation Files
Multimedia Multipathway Simulation Processor
Data Processor II
Met Database
Global Results Files
Exit Level Processor I
Risk Summary Output File
Exit Level Processor II
Protective Summary Output File
Risk Visualization Processor
Chemical Properties Processor
Chemical Properties Database
Distribution Statistics
Site Definition
Computational Optimization
Multimedia Multipathway Simulation
Exit Level
Key:
Processor
Database
Data Files
Shading indicates components that are designed into the system yet will not be functional by Oct. 31, 1999.

Figure 1.1 Overview of the FRAMES-HWIR Technology Software System

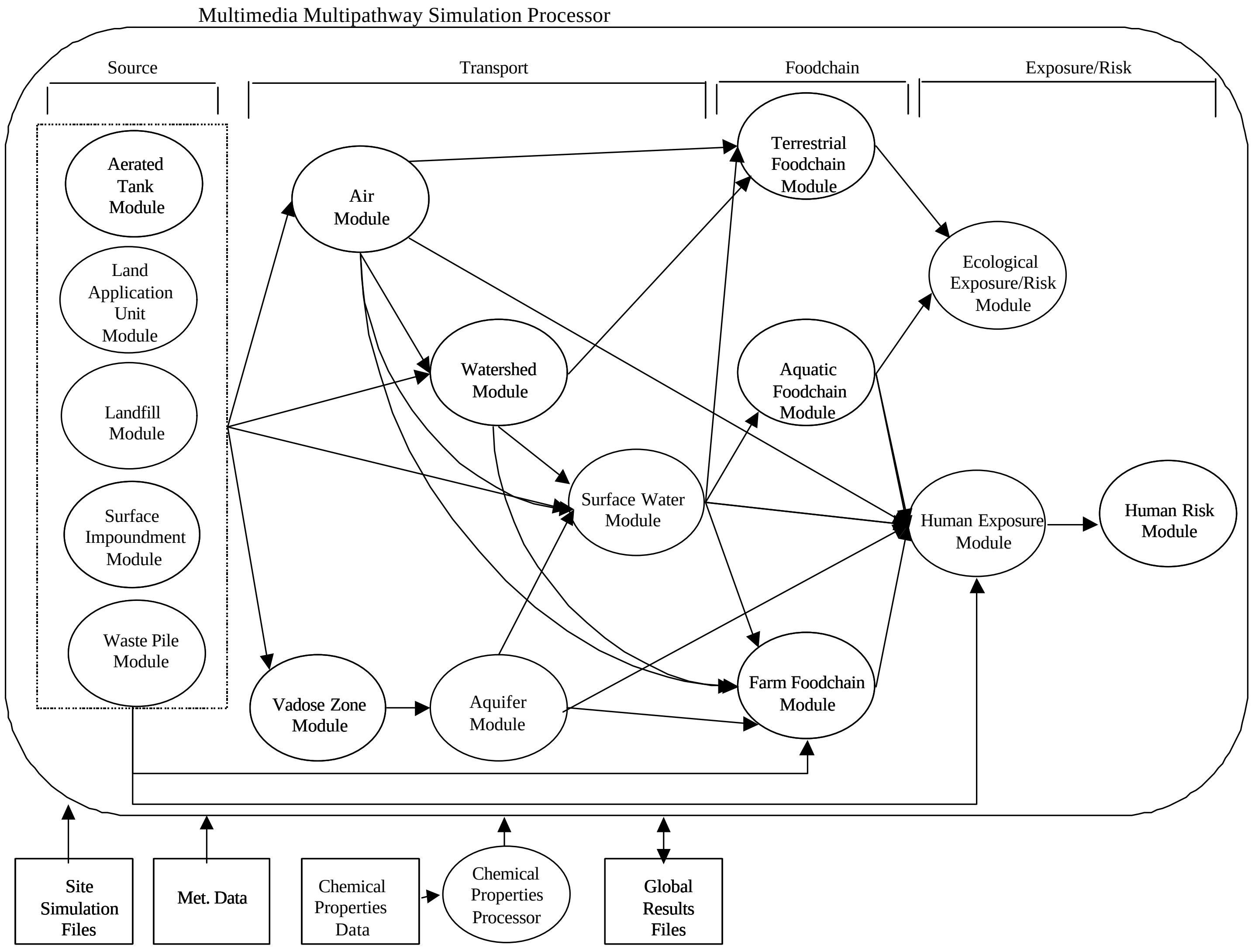


Figure 1.2 Modules in the Multimedia Multipathway Simulation Processor

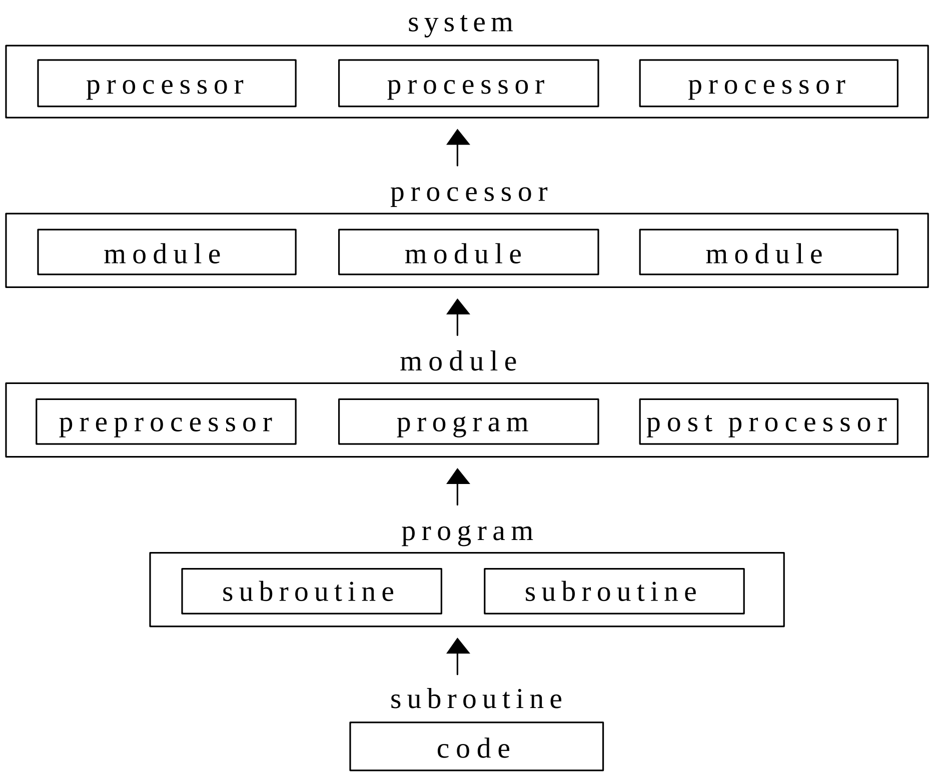


Figure 1.3 Levels of Testing

2.0 Software Development Guidelines

Software developers are expected to design and develop their respective components (module or processor) of the FRAMES-HWIR Technology Software System such that the component functions for its intended purpose and meets the needs of the overall system. Key elements for design and development are reading and writing data, error handling, hardware expectations, and system expectations. The following information breaks down each of these categories into specific guidelines for developers.

2.1 Reading and Writing Data

All components are required to read and write data that are collected into Data Groups. As the name implies, a Data Group is a collection of related variables within a database or data file. For example, the information regarding site layout would be collected into a Site Layout Data Group.

The modules within the MMSP will read specified Site Simulation Files and read and write Global Results Files, if appropriate. Software developers who are developing modules for the MMSP are expected to design modules to reformat and reorder data to meet the specific needs of the module. If pre- or post-processors are created for a module, then a "batch" file would also be required as part of the module. This "batch" file would serve as the single entry point to execute that module.

Modules are expected to read respective Data Groups from the Site Simulation Files and Global Results Files using the FRAMES-HWIR Technology Software System data specifications. The module is expected to read these Data Groups using the subroutines in the input/output dynamic link library (HWIRIO.DLL). Most of the information a module will need to function will be located in a Site Simulation Files or Global Results Files Data Group. However, some modules will need information from the Meteorological Database or Chemical Database (see Figure 1.1). If a module needs information from these two databases, that module is expected to read the appropriate file using methods designed and implemented by EPA for the Meteorological Database or by PNNL for the Chemical Database.

Modules are expected to write respective Data Groups to the Global Results Files using the FRAMES-HWIR Technology Software System data specifications and subroutines in the HWIRIO.DLL. The data specifications of the Global Results Files will dictate all the output variables that are expected to be written by a module. For both modules and processors, the writing of temporary files is acceptable, but the files should be deleted before exiting the component, such that the remaining files match the appropriate storage level given to the module as a call argument.

2.2 Error Handling

The modules that are subcomponents of the MMSP will, in general, have certain responsibilities. Errors and warnings will be reported to the MMSP module execution manager via an output file that will be created using the HWIRIO.DLL. The MMSP module execution manager will echo modules' warnings and errors to the SUI as well as any warnings and errors that occur within the module execution manager itself. If an error is produced by a processor, the processor should report the error directly to the SUI using the HWIRIO.DLL, which will then close the error file and allow the SUI to terminate execution of the processor. Because it may be difficult to change each module to correctly detect errors or crashes, each module is expected to create an error file in flat-ASCII (American Standard Code for Information Interchange) format. This file is created at the start of execution and then deleted at the end of execution, provided that there are no errors produced. If an error is produced, then the module should report the error using the HWIRIO.DLL, which will then close the error file and allow the MMSP module execution manager to terminate execution of the module and itself. The opening/creation and closing/deletion of this file is handled automatically with the OpenGroups and CloseGroups Subroutines provided in the HWIRIO.DLL.

Each module is also expected to create a warning file in flat-ASCII format at the start of execution and delete it at the end of execution provided that there are no warnings produced. If a warning is produced, then the module should write a warning message to the warning file using the HWIRIO.DLL and continue processing. Processors are also expected to create warnings files in the same manner. The opening/creation and closing/deletion of this file are handled automatically with the OpenGroups and CloseGroups Subroutines provided in the HWIRIO.DLL.

2.3 Hardware System Expectations

The FRAMES-HWIR Technology Software System is a PC-based set of executables. Modules and processors are expected to consume less than the full allotment of the memory, disk space, and time resources available to conduct the HWIR assessment. The current resource assumptions for a module or a processor are 16 Mb of RAM, 250 Mb of disk space, and an execution time on the order of 1 second on a stand-alone PC (assuming a Pentium [586] 200 MHZ PC-based machine).

Modules should limit data storage. Each module will be assigned a data storage level via a call argument from the module execution manager. This level determines the volume of data that can be saved. The module is responsible for generating no more data than can fill the respective data storage level. Level 1 stores minimal data; level 5 stores all data generated. If a module is given a data-storage level between 1 and 5, then the module is responsible for determining what files are to be saved. If a file is left in temporary space, then that file will be copied to permanent space. The writing of temporary files is acceptable, but these files should be deleted before the module exits, such that the remaining files match the appropriate storage level given to the module.

Software developers are expected to use only C++ and FORTRAN 77 (or later version) programming languages for coding all components, including processors, models, and pre- and post-processors, to use the features of the HWIRIO.DLL subroutines. If any other language is used, it is the responsibility of that component's developer to produce equivalent subroutines for file interfacing.

2.4 System-Level Expectations

It is the component developer's responsibility to ensure that the component meets the specifications outlined in the document entitled *Documentation for the FRAMES-HWIR Technology Software System, Volume 8: Specifications* (Castleton et al. 1998) and can interface with other portions of the system. For instance, if a shared program, such as the HWIRIO.DLL, does not meet all the modules's input/output needs, the module can be modified by adding a pre- or post-processor. Two key shared programs are the spatial and temporal integrator DLL and the HWIRIO.DLL.

If pre- or post-processors are developed for a model, module developers are expected to design, implement, and test these processors. Design and implementation of the MMSP module execution manager stops at the specified Site Simulation Files and Global Results Files. The module developers are expected to reformat and order data to meet the specific needs of their module.

3.0 Quality Assurance Guidelines

All software for the FRAMES-HWIR Technology Software System will be developed using an appropriate approach to quality assurance and documentation. Quality assurance procedures should be followed to ensure adequate communication of requirements and design solutions between software development efforts. Module developers should work with developers of related modules to ensure consistency of assumptions and data transfer between media.

The documentation required for each component of the software system (processor or module) consists of requirements documentation (Section 3.1), design and specifications documentation (Section 3.2), testing documentation (Section 3.3), mathematical formulations documentation (Section 3.4), and user's guidance documentation (Section 3.5). This documentation will help other members of the system development team understand the needs of the module or processor and communicate those needs to other team members. This information does not need to be in a formal report; however, it should be complete and organized such that a separate modeling team can interpret the module's input requirements and outputs produced. The following sections describe this documentation in more detail.

3.1 Software Requirements Documentation

Requirements documentation describes the characteristics and behaviors that a piece of software must possess to function adequately for its intended purpose. For example, a requirement for the FRAMES-HWIR Technology Software System is that all components must pass appropriate information for other components and ultimately the user.

Besides describing the requirements for a component, requirements documentation should also provide information showing that the approach to be taken is scientifically defensible. Each processor or module must have a description of the basis for the scientific approach planned. This basis should be directly related to the HWIR Assessment Strategy.

Requirements documentation 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.

3.2 Software Design and Specifications Documentation

The design and specifications documentation is a description of how a component will meet its requirements. The design portion of this description includes the design layout, often in the form of a flowchart. Design elements can be distinguished from requirements in that there are alternatives; in other words, a requirement is fixed, but a variety of design elements can be developed to meet it. For example, for the requirement of intercommunication mentioned above, components may use a shared program, develop pre- or post-processors, or, in the case of previously developed code, revise the code to ensure appropriate communication across the system. Which of these alternatives is chosen becomes the design element used by the component developer.

Specifications documentation describes in detail how information or data are stored or transferred between components created by separate developers. While design information is often no more detailed than a flowchart and good description of the elements, specifications go into additional detail with listings of example code and file naming conventions.

3.3 Software Testing Documentation

Each component of the FRAMES-HWIR Technology Software System should have a test plan that has been implemented with results documented. An outline of such a test plan is provided in Appendix A. Details related to testing are provided in Section 4.0.

3.4 Mathematical Formulations Documentation

The mathematical formulations documentation should encompass a description of all scientific algorithms used in the module or processor with a description of variables, defaults, and explanation of how the algorithms have been incorporated. The mathematical formulations documentation is sometimes associated with the user's guidance documentation, design documentation, or requirements documentation. In any form, however, the formulations should be compiled in a complete and logical format to aid in understanding the component's behavior.

Another piece of information in the mathematical formulations documentation is the confirmation of parameter relationships. For example,

- Total porosity is derived from bulk density and particle density.
- The partitioned mass in a source-term and in a system is directly related back to the total mass in the system.
- Rectangular area is derived from length and width.
- Rectangular paralleloped is derived from length, width, and thickness.
- Dose is derived from intake and concentration.

This documentation ensures that situations and/or variables that impact other variables are accounted for in a realistic manner. For instance, a module should provide internal checks for consistency. For example,

- A humid environmental setting would not necessarily be appropriate in an arid region.
- Modifications of soil type should impact the percent sand, silt, and clay; Kd; hydraulic conductivity; pore-water velocity; moisture content; porosity and bulk density; sorption capabilities; kinetics; erosion capabilities; etc. For example, Kd would not be expected to decrease when the organic matter and clay contents increase.
- The retardation factor and pore-water velocity in the saturated system maintain a consistent relationship with respect to effective porosity. For example, if nonflowing voids are assumed to be present, then the effective porosity should be in both expressions.
- Mountainous regions should contain topographic relief.

3.5 User’s Guidance Documentation

The user’s guidance for a module or processor should include information on the appropriate use of the component, including an overview of technical/science aspects of the module or processor. These technical/science aspects include goals/purpose, assumptions/assertions, and technical capability/limitations. This documentation should also provide information on ranges for input and output variables of the component. This information will help other team members to ensure that component needs are met by the wider system and that system checks can be conducted to ensure that values for variables are within appropriate ranges and physically make sense. For example,

C The average Darcy infiltration rate through a landfill is less than the average precipitation rate.

C Total and effective porosities, moisture content, and individual risk should not be greater than unity (by fraction).

C Latitudes are between 20 and 80, and longitudes are between 60 and 180.

• Moisture content (by fraction) cannot be greater than the porosity (by fraction).

C Source bulk density cannot be greater than the particle density.

C Soil-type classification matches the U.S. Department of Agriculture’s soil textural triangle. For example, a loamy sand cannot be composed of 20% sand, 65% silt, and 15% clay (this mixture is classified as a silty loam). Soil types designated as “sand” must be composed of over 80% sand.

C Bulk density should generally be between 1 and 3 g/cc.

C Particle density is typically 2.65 g/cc, although there is an acceptable range.

C Total mass released by the source term through all release mechanisms does not exceed the total mass originally in the source.

C Mass flux rate is cross-correlated back to concentration to ensure that the concentration is well below the solubility limit and density of the constituent.

4.0 Testing Components of the FRAMES-HWIR Technology Software System

The FRAMES-HWIR Technology Software System is a large software system with a relatively high level of complexity. Because of its object-oriented design, however, it can be broken down into less complex components that are easier to test. The testing strategy/protocol being used for this system consists of 1) development teams preparing test plans for the components they develop, 2) development teams implementing the test plan, correcting any problems found, and documenting the testing results, and 3) an external independent tester conducting final acceptance testing.

The following subsections describe the levels at which testing can be conducted (Section 4.1), the objectives of testing (Section 4.2), the relationship between the testing and development process (Section 4.3), and the appropriate testing process for components of the FRAMES-HWIR Technology Software System (program, module, processor, or system; Section 4.4).

4.1 Levels of Testing

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. This is the primary testing of the code; therefore, each component developer must ensure that unit testing of programs is thorough and well 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 system (also called integration testing), as well as the overall performance of the system. Issues that arise during system testing are addressed by fine-tuning how individual components run so that they work together properly as a team. Each module or processor developer is responsible for ensuring that the individual components of his/her module or processor communicate and work properly with each other. Each developer is also responsible for ensuring that his/her module or processor communicates properly with the wider system in which his/her component is embedded. The use of shared routines, such as the HWIRIO.DLL and the spatial and temporal integrator DLL, may be helpful in achieving this goal.

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.

4.2 Objectives of Testing

Three main objectives are associated with testing:

1. Uncover errors in requirements, design, and programming. Software developers may have to revise requirements documentation, design documentation, and program code implementation to resolve issues that are found.

2. Identify or confirm limitations of the system. Software developers will need to document these limitations (perhaps in a revision to the design documentation) for future reference. Software developers will also have to eliminate limitations that inhibit the required functionality of the software.

3. Build confidence in the capabilities of the system. By the end of the testing period, the tester should be confident that the software meets its requirements. Testers at all levels shall keep documentation to support this conclusion.

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.

To ensure that testing is completed satisfactorily, testing procedures should be planned thoroughly in advance. An annotated outline of the contents of a test plan is shown in Appendix A. Adhering to the testing procedures developed is equally important. If the test plan is not followed, it will be difficult to determine what was tested, why it was tested, and if the testing was adequate to show that the software meets its requirements.

Quality testing requires an in-depth knowledge of the system being tested. This knowledge includes understanding 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. Finding testers with in-depth knowledge of the system, who are also independent from the development team, can be challenging. This apparent contradiction is addressed by the two-stage testing process of development team testing followed by external independent testing, as described below.

4.3 Relationship Between the Testing Process and the Development Process

The relationship between the testing process and the development process is shown in Figure 4.1. As this figure shows, the complete testing process runs parallel to the development process. The testing process begins by identifying requirements, which are "tested" through reviews. The requirements are reviewed for testability and clarity and to ensure that they will meet the needs of the user. The software design is also reviewed for testability and clarity, as well as to ensure that all requirements are addressed.

Ideally, once the requirements and design documentation are complete, program coding and test plan preparation can begin simultaneously. A test plan addressing the functional requirements of the software should be able to be written using only the requirements and design documentation. If this is not possible, then the requirements and/or design documentation are incomplete. When both program coding (including any informal testing conducted by the programmers) and the test plan are complete, formal testing begins by implementing the test plan. If the test plan is not prepared simultaneously to program coding and instead is prepared after coding is complete, then software testing is held up, and time is lost.

In practice, however, test plan preparation and program coding may not begin simultaneously. Often issues that arise during coding cause changes in the design and possibly even the requirements. Therefore, it may be desirable in some cases to delay plan preparation until coding is far enough along to provide confidence that the design is final.

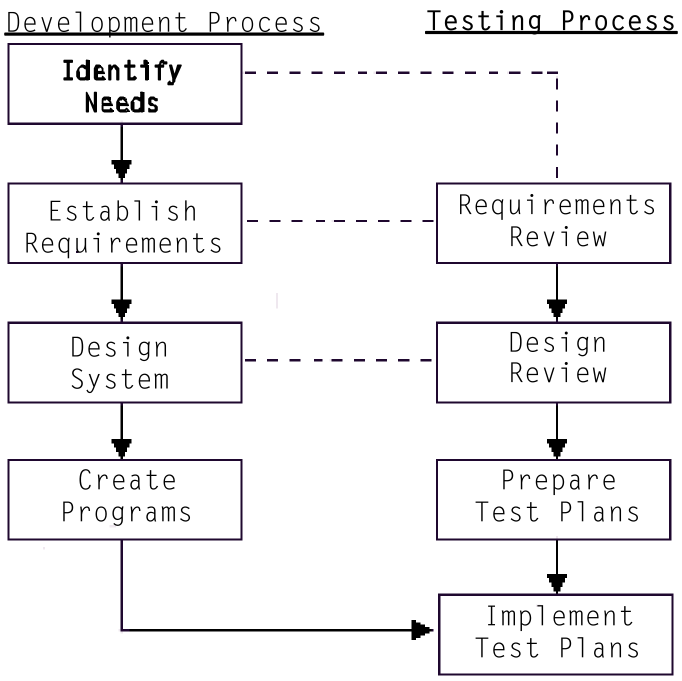


Figure 4.1 Parallel Development Process

4.4 Software Testing Process

All components (modules, processors, and the system itself) being developed for the FRAMES-HWIR Technology Software System will be tested internally by that component's developers (or by someone internal to the developer's organization, but not necessarily directly involved in program coding) and then externally by an independent party. The steps involved in each testing process are described below.

4.4.1 Internal Testing Process

The steps involved in the internal testing process are as follows:

1. The developer of the software performs program coding and informally tests to ensure that the software is operational, and no obvious errors exist.

2. The internal tester designs test cases and a test plan based on the requirements document and the design document related to the software.

3. The developer delivers the software to the internal tester.

4. The internal tester implements the test plan and evaluates the results.

5. If the results are acceptable, the tests results are documented, and internal testing is finished. If the results are unacceptable, the internal tester reports problems to the software developer.

6. A decision will be made whether to fix the problems. If the problems will not be fixed then testing is complete. If a problem does require fixing, then the software developer will modify the code, and the internal tester will rerun the test cases.

7. This process is repeated until all test results are acceptable, or it is decided that the remaining problems do not require fixing.

Figure 4.2 depicts this process in a flow diagram. Once the internal testing is complete, software developers will provide the component executables, the source code, the test plan, and testing results documentation to an external independent test group for verification.

4.4.2 External Independent Testing Process

The steps involved in the external independent testing process are as follows:

1. The software developer will deliver the module executable, source code, test plan, and testing results documentation to an external independent test group.
2. The external independent test group critically reviews the developer's test plan for completeness.
3. The external independent test group documents additional tests to be conducted (if necessary).

4. The external independent test group (re)executes all tests and any additional tests using the delivered program executables.

5. Working iteratively, the external independent test group documents any errors found and communicates with the software developer. The software developer corrects code and redelivers the executable and source code to the external independent test group to continue testing.

6. The external independent test group recompiles source code(s) and rebuilds the executable files.

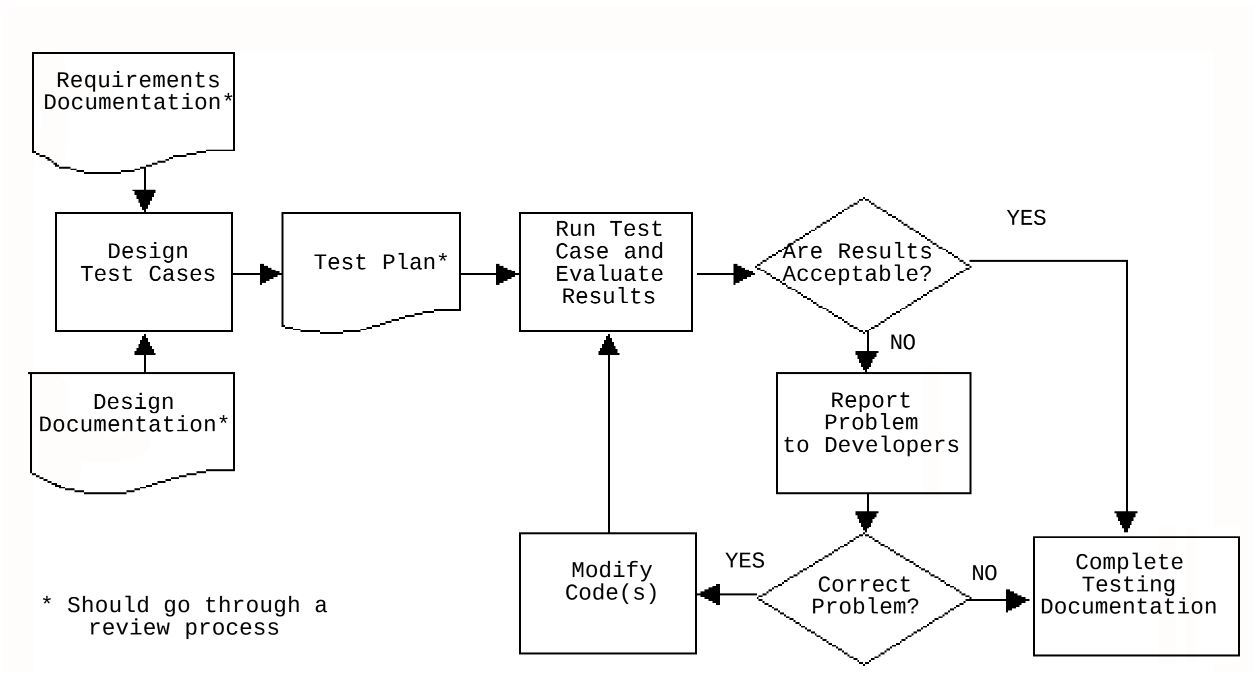

Requirements Documentation*
Design Test Cases
Test Plan*
Run Test Case and Evaluate Results
Are Results Acceptable?
YES
NO
Design Documentation*
Report Problem to Developers
Modify Code(s)
YES
Correct Problem?
NO
Complete Testing Documentation
* Should go through a review process

Figure 4.2 Internal Testing Process

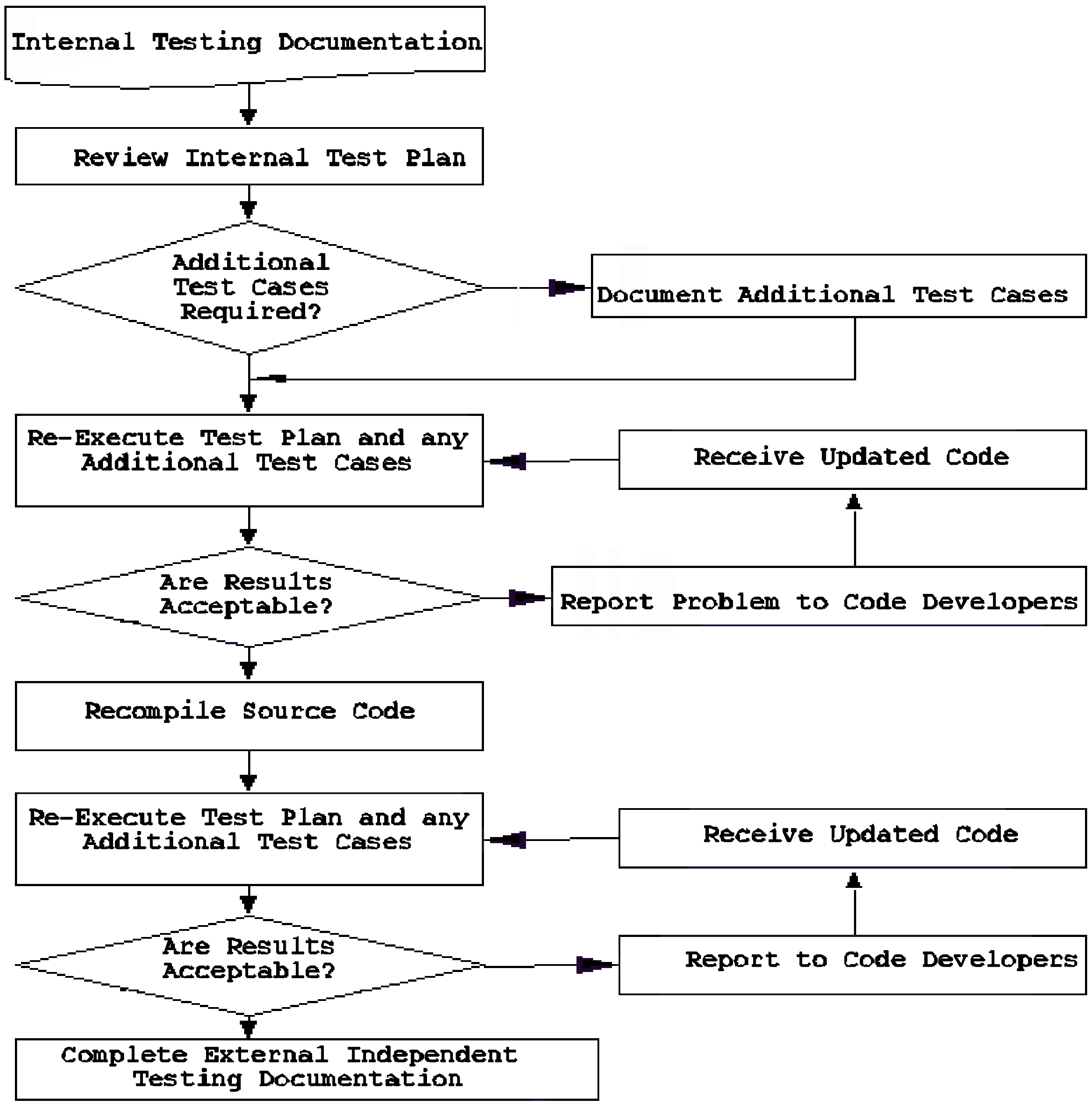
7. The external independent test group (re)executes the tests conducted under Step 4 using the new executable files. The iteration with the module developer, as described in Step 5, is repeated until all test results are acceptable, or it is decided that the remaining problems do not require fixing.

Figure 4.3 depicts this process in a flow diagram. Once this testing is complete, the independent external test group completes a package of testing documentation for client review.

4.4.3 Preparation of a Test Plan

The programmer usually conducts informal unit testing of subroutines and programs within software components as part of code development. While coding is proceeding, the tester will prepare a test plan describing exactly how that component will be tested. This test plan should include test cases for both unit testing and system testing of the programs composing that component. Appendix A contains an annotated outline describing the contents of a test plan. In general, a test plan should contain

C a listing of requirements for each program comprising the component, as well as any additional requirements for the software component as a whole (i.e., explain what each program is supposed to do, and what the component as a whole is supposed to do)



Internal Testing Documentation
Review Internal Test Plan
Additional Test Cases Required?
Document Additional Test Cases
Re-Execute Test Plan and any Additional Test Cases
Receive Updated Code
Are Results Acceptable?
Report Problem to Code Developers
Recompile Source Code
Re-Execute Test Plan and any Additional Test Cases
Receive Updated Code
Are Results Acceptable?
Report to Code Developers
Complete External Independent Testing Documentation

Figure 4.3 External Independent Testing Process

- detailed descriptions of planned test cases, including all input data and instructions necessary to execute each test
- the expected results for each case to provide some criteria used to determine whether the test was successful.

Test cases will be developed using three approaches: black box, white box, and a consideration of extremes and boundary conditions.

First, test cases for a software component are designed from a black-box approach, that is, by knowing the exact input requirements, by knowing the solution technique implemented by the component, but not knowing how the technique was coded (i.e., programmed), and by knowing the exact output requirements. These cases will address all of the requirements outlined for that component. A good approach to designing test cases is to start with the most simple case possible and gradually add complexity until the most complex case is derived.

Once black box testing is complete, a white box approach will be employed (particularly for unit testing of programs) by identifying lines or blocks of code not ever executed by the black box test cases. Additional test cases will then be developed such that all lines of code will be executed at least once during testing. The results of black box and white box testing establish the capabilities of the system.

Finally, drawing on the tester's understanding of both the component and the solution technique implemented, a set of input-data extremes as well as boundary conditions will be identified, and a set of test cases will be designed to test these conditions. This part of the test-case design requires the most creativity and insight on the part of the tester, and the results of these tests establish the limitations of the system.

Ideally, the test plan will be completed and then implemented. In reality, however, this process may proceed in three phases. First, the black box cases are prepared and then implemented. During implementation, the degree of code coverage can be determined, after which white box cases can be added to the test plan and then implemented. Once white box testing is complete, the tester will be in the best position to identify extremes and boundary conditions to be tested. These cases can then be developed, added to the test plan, and then implemented. Once all testing is complete, the final testing results will be documented. The final results documentation will address the capabilities and limitations of the software component.

5.0 References and Bibliography

5.1 References

Castleton, K. J., M. A. Pelton, B. L. Hoopes, and G. Whelan. 1998. *Documentation for the FRAMES-HWIR Technology Software System, Volume 8: Specifications*. PNNL-11914, Volume 8, Pacific Northwest National Laboratory, Richland, Washington.

5.2 Bibliography

Hetzel, W. 1984. *The Complete Guide to Software Testing*. QED Information Sciences, Inc., Wellesley, Massachusetts.

5.3 Companion Documents

Volume 1: Documentation for the FRAMES-HWIR Technology Software System, Overview of the FRAMES-HWIR Technology Software System, PNNL-11914, Volume 1, Pacific Northwest National Laboratory, Richland, Washington.

Volume 2: Documentation for the FRAMES-HWIR Technology Software System, System User Interface, PNNL-11914, Volume 2, Pacific Northwest National Laboratory, Richland, Washington.

Volume 3: Documentation for the FRAMES-HWIR Technology Software System, Distribution Statistics Processor, published by Tetra Tech, Lafayette, California.

Volume 4: Documentation for the FRAMES-HWIR Technology Software System, Site Definition Processor, PNNL-11914, Volume 4, Pacific Northwest National Laboratory, Richland, Washington.

Volume 5: Documentation for the FRAMES-HWIR Technology Software System, Computational Optimization Processor, published by Tetra Tech, Lafayette, California.

Volume 6: Documentation for the FRAMES-HWIR Technology Software System, Multimedia Multipathway Simulation Processor, PNNL-11914, Volume 6, Pacific Northwest National Laboratory, Richland, Washington.

Volume 7: Documentation for the FRAMES-HWIR Technology Software System, Exit Level Processor, PNNL-11914, Volume 7, Pacific Northwest National Laboratory, Richland, Washington.

Volume 8: Documentation for the FRAMES-HWIR Technology Software System, Specifications, PNNL-11914, Volume 8, Pacific Northwest National Laboratory, Richland, Washington.

Volume 9: Software Development and Testing Strategies. 1998. PNNL-11914, Vol. 9, Pacific Northwest National Laboratory, Richland, Washington.

Volume 10: Documentation for the FRAMES-HWIR Technology Software System, Software Development Kit, PNNL-11914, Volume 10, Pacific Northwest National Laboratory, Richland, Washington.

Volume 11: Documentation for the FRAMES-HWIR Technology Software System, User's Guidance, PNNL-11914, Volume 11, Pacific Northwest National Laboratory, Richland, Washington.

Volume 12: Documentation for the FRAMES-HWIR Technology Software System, Dictionary of Terms, Acronyms, and Abbreviations, PNNL-11914, Volume 12, Pacific Northwest National Laboratory, Richland, Washington.

Volume 13: Chemical Properties Processor Documentation. 1998. PNNL-11914, Vol. 13, Pacific Northwest National Laboratory, Richland, Washington.

Volume 14: Site Layout Processor Documentation. 1998. PNNL-11914, Vol. 14, Pacific Northwest National Laboratory, Richland, Washington.

Volume 15: Risk Visualization Tool Documentation. 1998. PNNL-11914, Vol. 15, Pacific Northwest National Laboratory, Richland, Washington.

Quality Assurance Program Document

Gelston, G. M., R. E. Lundgren, J. P. McDonald, and B. L. Hoopes. 1998. *An Approach to Ensuring Quality in Environmental Software*. PNNL-11880, Pacific Northwest National Laboratory, Richland, Washington.

Additional References

Office of Civilian Radioactive Waste Management (OCRWM). 1995. *Quality Assurance Requirements and Description, Supplement I, 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.

Marin, C., and Z. Saleem. 1997. *A Preliminary Framework for Finite-Source Multimedia, Multipathway and Multireceptor Risk Assessment (3MRA)*. Draft, October 1997, U.S. Environmental Protection Agency, Office of Solid Waste, Washington, D.C.

Appendix A

Test Plan -- Annotated Outline

Appendix A
Test Plan – Annotated Outline

1.0 Background and Scope

A brief description of the module to be tested and the codes that comprise the module (i.e., core program or model, module user interface, and pre-/post-processors). This section should also describe the depth of testing (i.e., white box, black box, or both) and the reason for testing to that level.

2.0 Requirements

2.n Model/Module Name

List the requirements and number each requirement. Show a table (see example below) 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

3.n Test Case Name

3.n.1 Description and Rationale

A brief description of the case and the rationale for selecting this case. The intent here is to answer the question, "why was this case selected?"

3.n.2 Input Data

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 as is required by the component being tested. Input data common to more than one case can be listed once (say, in Appendix B) and referred to by a data set name (e.g., Local Climatological Data Summary for Fresno, CA). By listing all necessary input data in the test plan, the tester does not have to track down other references for input data.

3.n.3 Expected Results

Describe the expected results of the case. The intent here is to provide acceptance criteria for deciding if the test was successful. Comparisons to other verified models, monitored or sampled data, known analytical solutions, or hand/spreadsheet computations can be used for this analysis. The more specific the expected results are described, the better. For an atmospheric modeling case, for example, statements like “concentrations should be highest to the north and lowest to the southeast” are helpful. However, one or more specific hand-calculated (or spreadsheet-computed) concentrations are better. The detail in which the expected results are described depends on the difficulty of computing specific values. Hand-calculated concentrations for all distances and direction sectors in an atmospheric case is probably not feasible (given time and budget constraints) and may not be desired (if the program computes concentrations correctly at one or two points, it may be reasonable to assume it is working correctly for all points).

3.n.4 Special Procedures

List any special procedures necessary to execute the test (i.e., deviations from the general procedure explained in Appendix C). Specific information concerning codes, compilers, and linking protocols should be specified. Codes should not use specialized options that are unique to a particular compiler. Codes will be delivered to independent testers for recompilation, relinking, etc.

4.0 References

Appendix A - Expected Results Documentation

Hand or spreadsheet computations of expected results are documented here. For each test case, the intent is to show how the expected results were determined from the input data. This information must be presented in a form suitable for a reader to duplicate the computations.

Appendix B - Input Data Sets

List the common input Data Sets here. Be sure to use the same units as is required by the component being tested.

Appendix C - General Procedure for Test Case Implementation

Describe how someone would set up and run a test case for this module. If the tester is expected to populate input files by hand (i.e., not using an interface program), then the file format specifications need to be provided. If file format specifications are described in a separate document, then a reference to thatdocument is sufficient if it is delivered along with the test plan. This description should be complete enough for an independent party to conduct the test.