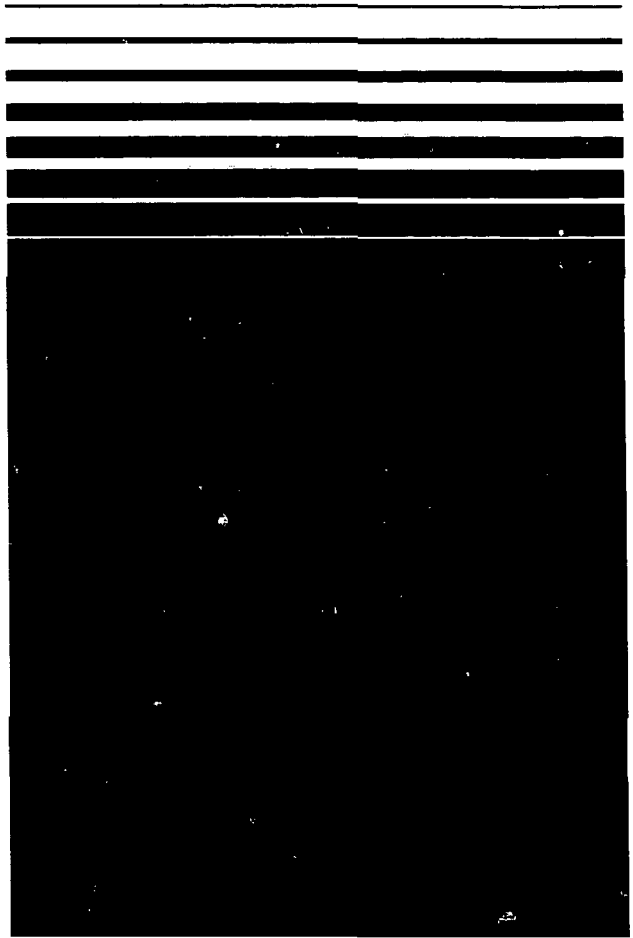
Report Rapport

Atomic Energy Control Board
Commission de contrôle de l'énergie atomique

VERIFICATION AND VALIDATION FOR WASTE DISPOSAL MODELS

by

INTERA Technologies Ltd.
Ottawa, Ontario



Atomic Energy Control Board
P.O. Box 1046
Ottawa, Canada
K1P 5S9

Commission de contrôle de l'énergie atomique
C.P. 1046
Ottawa, Canada
K1P 5S9

INFO-0246

VERIFICATION AND VALIDATION FOR WASTE DISPOSAL MODELS

by

INTERA Technologies Ltd.
Ottawa, Ontario

A research report prepared for the
Atomic Energy Control Board
Ottawa, Canada

Project no. 5.101.1

July 1987

Canada

Research report

VERIFICATION AND VALIDATION FOR WASTE DISPOSAL MODELS

ABSTRACT

A set of evaluation criteria has been developed to assess the suitability of current verification and validation techniques for waste disposal methods. A survey of current practices and techniques was undertaken and evaluated using these criteria with the items most relevant to waste disposal models being identified. Recommendations regarding the most suitable verification and validation practices for nuclear waste disposal modelling software have been made.

RÉSUMÉ

Le présent rapport fait état de la série de critères d'évaluation qui a été mise au point pour juger de la pertinence des techniques de vérification et de validation actuelles des méthodes d'évacuation des déchets. Les pratiques et techniques actuelles ont été examinées et évaluées selon ces critères et les éléments les plus importants par rapport aux modèles d'évacuation des déchets ont été indiqués. Des recommandations relatives aux pratiques de vérification et de validation les plus appropriées ont été faites pour les logiciels de modélisation de l'évacuation des déchets.

TABLE OF CONTENTS

LIST OF TABLES

1. INTRODUCTION

1.1 BACKGROUND

As part of its mandate, the Atomic Energy Control Board (AECB) is required to evaluate an applicant's performance and safety analyses during the licensing process for radioactive waste disposal sites. Due to the time scale over which the waste will remain a concern and the complex nature of the coupled processes likely to determine release rates, the use of models to support such performance and safety analyses has become widespread. To evaluate what can be done to ensure that the results provided by the model are both accurate and represent the physical processes, INTERA Technologies Ltd. have been retained by AECB to evaluate verification and validation techniques for waste disposal models.

The vast majority of the waste disposal programs currently in use were developed relatively recently (over the last 10 - 15 years) predominately by scientists or engineers who had little appreciation for the software engineering aspects of the programs. For these reasons, the vast majority of these programs were written in FORTRAN for mainframe (Cyber 176 or equivalent) computers although more recently some of this software has been written in more modern languages (e.g., PASCAL) for both micro and mini based computing systems.

The processes controlling/influencing the release of radionuclides from a nuclear waste repository are numerous and varied and include:

- groundwater transport;
- geochemical reactions;
- heat transport;
- stress-strain reactions;

- surface water transport;
- bioaccumulation.

Each of these transport areas are comprised of a number of mechanisms which may affect the overall rate of radionuclide movement. Groundwater transport, for example, should consider convection, dispersion, diffusion as well as radionuclide decay/generation and adsorption. The complexity of the problem may also be increased by the coupled nature of the proceses. Groundwater transport, for example, will be influenced by the density and viscosity changes caused by temperature variation. Tsang (1985) presents a list of the coupled processes that may be encountered in the vicinity of a nuclear waste repository (see Table 1.1).

The models constructed to represent these physical processes may take many forms. The models may be physically-based, that is based on known or well established laws or scientific principles, or empirically based, that is based on correlations established using observations. Both types of models have certain limitations or restrictions that affect their use.

Whereas empirical models may be computationally simple, they rely on an extensive data base to formulate the correlations required over the range of observations. Physically based models are typically formulated using one or more fundamental governing equations. These equations may be complex being represented by several partial differential equations in multi-dimensions. Several approaches may be employed to solve such equations. In cases where the problem may be reduced to one or two-dimensions and the system further simplified (e.g., through geometry, temporal conditions) an analytical solution to the partial differential equation may be determined. A number of simple groundwater transport models have been constructed using this approach (Pigford et al., 1980). In the general case where geometry

Table 1.1 List of Coupled Processes (from Tsang, 1985)

- Thermochemical (TC)
  - Thermal Diffusion
  - Phase Changes--Equation of State of Solids
  - Solid Solution
  - Metastable Phases
- Thermohydrological (TH)
  - Convection Currents (1 or 2 phases) including Buoyancy Flow
  - Phase Changes and Interference
  - Thermophysical Property Changes
  - Thermal Osmosis
  - Gas Diffusion:
    - Binary
    - Knudsen
    - Thermal
    - Coupled
  - Capillary-Adsorption
- Thermomechanical (TM)
  - Induced Cracking
  - Fracture Deformation
  - Thermal Spalling
  - Thermal Creep
  - Thermal Expansion
- Hydrochemical (HC)
  - Speciation
  - Complexation
  - Solution
  - Deposition
  - Sorption/Desorption
  - Redox Reactions
  - Hydrolysis
  - Acid-Base Reactions
  - Diffusion
  - Chemical Osmosis and Ultrafiltration
  - Isotopic Exchange
  - Coprecipitation
- Hydromechanical (HM)
  - Hydraulic Fracturing
  - Pore Pressure Change
  - Hydraulic Erosion of Fractures
  - Sedimentation of Particles
  - Shear Effect Causing Abrasion
  - Variation of Fracture Apertures
  - Filtration of Particles
  - Ultrafiltration

Table 1.1 List of Coupled Processes (from Tsang, 1985) (cont'd)

VI. Mechanical-Chemical (MC)

These processes will be modified by hydrological and thermal effects, so they are included in processes VII and VIII below.

VII. Thermohydrochemical (THC)

Solution/Precipitation
Time-Dependent Solution and Precipitation
Fluid Transport by Osmotic Effect
Chemical Transport in Gas Phase
Partition between Gas and Solid
Particle Transport (Colloids)
Equation of State
Thermal Diffusion (Soret Effect)
Thermal Osmosis

VIII. Thermomechanical-Chemical (TMC)

Phase Change in Mineral Phases
Dehydration
Creep
Hydration and Swelling

IX. Thermohydromechanical (THM)

Hydraulic Fracturing
Triggering of Latent Seismicity
  Stress Redistribution
  Pore Pressure
Opening and Closing of Joints
  Stress Redistribution
  Thermal Coupling
  Pore Pressure
Spalling
Change of Strength
  Stress Corrosion
  Hydrolitic Weakening
Hydration

X. Hydromechanical-Chemical (HMC)

These processes were eliminated from consideration because without thermal effects the mass transport is not sufficient to change the geometry except for low temperature precipitation, stress corrosion, and ion exchange producing swelling (e.g. Na for Ca in smectites) at approximately 50°C or below.

XI. Thermohydromechanical-Chemical (THMC)

Piping--Selected Dissolution and Tunnel Corrosion,
  Inhomogeneous Leaching
Precipitation
  In Fractures
  In Matrix
Hydrothermal Alteration of Rock
Heat Pipe Effect with Dissolution/Precipitation
Vertical Vapor-Liquid Cycling Near Canisters
Pressure Solution

and temporal conditions will not allow a simplification of the problem, a numerical approach must be employed to solve the equation. A wide variety of numerical techniques have been used to solve the various equations representing these processes, and include finite difference, finite element, method of characteristics and discrete parcel random walk techniques. Models employing these techniques offer advantages over analytical methods in terms of their generality, but certain trade-offs do exist with regards to their sophistication and ease of use.

The waste disposal models described to this point may all be classified as deterministic. That is, the basic objective is to calculate a single best estimate value without taking any of the possible variation in input parameters into account. Given that such variation does exist and owing to the significant environmental consequences associated with the results of such an analyses, increasing emphasis is being placed on the reliability of these results. One approach aimed at quantifying the uncertainty associated with such analyses is through the use of probabilistic models or modeling techniques. In some cases, these models use the basic deterministic model together with a probabilistic interface (e.g., sampling methodologies) to establish confidence levels on the output variables. Other probabilistic modeling approaches have included statistical techniques on the input data, finite order methods and particle motion methods. It is worth noting that while research and development into new modeling techniques/applications are ongoing in all areas of waste disposal models this is particularly true of the probabilistic area which has only recently become an area of interest.

1.2 OBJECTIVES

Given the increasing reliance of various waste disposal agencies on the use of computer models to assess various radioactive transport mechanisms and dose exposures to man, the need for ensuring

that the answers provided by the models are both correct and represent the physical process for which they are intended is of utmost importance. To assist AECB staff in assessing the verification and validation activities that have been performed for particular software models this study was commissioned with the following specific tasks:

1. Develop criteria to evaluate the suitability of various verification and validation standards, techniques and tools to waste disposal models.

2. Survey the available literature and assess state-of-the-art verification and validation standards and methods in the context of their applicability to waste disposal models using the criteria established in Task 1.

3. Document the findings in a report which includes:

   i) criteria developed in Task 1;
   ii) assessment of the verification and validation survey;
   iii) discussion of relevant verification and validation standards and approaches;
   iv) recommendations of verification and validation practices most appropriate to waste disposal models.

In summary, the overall objective of the study is to survey current verification and validation methods that are being (or could be) used for modeling software and to identify those methods best-suited to waste disposal models.

2. VERIFICATION

2.1 DEFINITION

The term verification does not appear to have a consistent and universal definition. The primary reason for this is associated with the fact that it is a very common word and is used by a wide variety of individuals including scientists, software engineers, etc. For the purpose of this study verification is defined as the process by which one becomes confident that the computer code correctly executes the calculations it is asserted to perform. In other words, a verified program is one that has translated a specified algorithm into coding correctly.

Verification is one of the steps involved in obtaining a benchmarked or baselined program. A benchmarked or baselined program is one that has been tested to a certain specification (verification and/or validation). The resulting program is typically assigned a unique version number and the program, test specifications and results are documented and archived. No modifications to this version of the program are allowed once it has been baselined unless a new version number is assigned. The benchmark of computer codes is a fundamental component of most formal QA programs.

2.2 TYPES OF ERRORS

There are four main types of errors that may affect the output provided by scientific software, and include:

1. language errors;
2. hardware errors;
3. logic errors;
4. numerical algorithm errors;
5. data input errors.

In many cases the division between these categories is not clear and some overlap may exist. The following sections discuss each of these error types, the most prominent kinds of errors and specific examples of each. It should be noted that this discussion considers only legitimate post-compiler errors. Syntax errors that would be identified by a compiler are not considered, neither are questionable or error prone constructs.

2.2.1 Language Errors

The following discussion deals primarily with source code errors although it should be noted that errors may also exist in compiler and operating system software. Errors found in computer software may be classified in one of the following categories:

1. Data declaration errors;
2. Data reference errors;
3. Interface errors;
4. Comparison errors;
5. Input/output errors;
6. Miscellaneous.

Data declaration errors would include such errors as mismatched variable type (e.g., INTEGER, REAL) or attribute (e.g., single versus double precision) occurring during variable initialization or definition. In many instances default variable typing associated with FORTRAN is responsible for these errors. Other data declaration errors may arise through the use of EQUIVALENCE statements (if inconsistent parameter types are used) or by over-initializing or defining an array (i.e., indices out of range). Failure to include arrays in either COMMON block or DIMENSION statements may result in these quantities as being classified as external functions. An execution error will occur if statement execution is attempted.

Some data declaration errors may also be classified as data reference errors especially in those instances where the error occurs in an initialization or data definition statement. Data reference errors may occur when an uninitialized or undefined variable is referenced or a subscript is out of range or the subscript is invalid.

Interface errors are associated with the transfer of variables (addresses) to various portions of the program and occur primarily in subroutine argument lists or common blocks. The errors arise if inconsistencies exist in the number or type of variables being transferred from, for example, the subroutine call statement to the subroutine itself.

Most comparison errors occur through the use of mismatched variables (either type or attribute) in a logical test (e.g., LT, GT, EQ). A similar problem may occur in mixed mode arithmetic where the use of INTEGER and REAL variables in the same statement may lead to unexpected results.

A variety of errors may be associated with the input or output (I/O) of data. This would include errors associated with format statements (e.g., inconsistent variable and format mask, improper use of scale factors, printer control characters); inconsistencies between file attributes and read statement (e.g., formatted versus unformatted files); incorrect file positioning (e.g., failure to REWIND); or improper use of I/O units (e.g., reserved units).

The five categories identified above typically account for over 90% of the software errors in compiled source code. The remaining errors are obscure and quite often are the hardest to locate. For example an error may result by typing the last letter of a variable in column 73. This would result in the creation of a new undefined variable being used to evaluate the expression. Other obscure errors are sometimes associated with limitations of the computer system, such as a limit on the number of units that can be used, or restrictions on the number of print columns. Additional examples of language errors are presented in Appendix A.

2.2.2 Hardware

Although a number of machine dependent factors have been identified as possibly influencing the accuracy of the results of a scientific program (Battye and Parsaei, 1980) the single most important and influential factor is the word-length (or number of bits used to represent data) and whether the storage system is fixed point or floating point format. Fixed point storage is used for integer values and arithmetic operations performed between these numbers is exact (i.e., no round-off) provided that the answer is not outside the range of integers that can be represented and that division is assumed to result in an even integer (i.e., no remainder). In a binary 32-bit system (INTEGER*4) the permissible range is approximately $\pm$ 2.1 x 10^9 while in a 16-bit system this decreases to $\pm$ 32,767.

Floating point representation is used for real numbers and is comprised of a sign bit, an integer exponent and an integer mantissa. Unlike fixed point systems, arithmetic operations are not necessarily exact. This results from the "normalizing" that takes place in the floating point system, where the bit patterns of the mantissa are shifted as far to the left as possible to preserve as much accuracy as possible. Unfortunately in instances where a large number and a smaller number are used in an arithmetic operation this process may result in the loss of the lower-order-bits of the smaller number. It is important to realize that this loss of accuracy (known as round-off error) is a function of the number of bits comprising the mantissa and not the exponent. Therefore although it may be possible to represent two numbers individually on a computer it may not be possible to represent their sum. For example on a binary bit machine a single precision (REAL*4) real variable can range from approximately 10^{76} to 10^{-79} although the machine accuracy is limited to 10^{-8}. If the error associated with such round-off errors is equally distributed (i.e., up versus down) then total round-off error should be on the order of $e_m(N)^{\frac{1}{2}}$ where e_m is the machine accuracy and N is

the number of arithmetic operations. However as pointed out by Press, et al., (1986) such unbiased accumulation of error rarely occurs and the actual total is more likely to be $e_m N$ or in some specific cases even higher.

The foregoing discussion has concentrated on how word-length may introduce round-off error due to machine accuracy limits. Word-length may also affect the transferability of programs to different machines. Some codes may be coded using a high degree of machine-dependency (e.g., assuming a specific word-length will be available) which can necessitate considerable reprogramming to convert to a different machine.

2.2.3 Logic Errors

There is virtually an unlimited number of logic errors that may exist in any program. The following discussion is meant to describe some of the most common ones.

Control flow errors are associated with the use of unanticipated paths within the program. This might include, for example, the transfer of control from a point outside a do-loop to one inside the loop, or the existence of infinite loops. Another commonly encountered error of this kind is the calculation of an invalid index for a computed GO TO statement. Some compilers may (depending on their error tracing capabilities) have options to identify some of these execution errors.

Other miscellaneous logic errors may be associated with the programmers unfamiliarity with the language and might include errors due to assuming a certain precedence in the execution of arithmetic operations, or the use of a reserved variable name for a use other than what it was intended. Other sometimes difficult to detect errors may be associated with variable array sizes that result in a portion of the memory being overwritten.

2.2.4 Numerical Algorithm Errors

Finally, no discussion of the possible errors that might be encountered in waste disposal models would be complete without some discussion of errors associated with the numerical techniques that form a significant subset of the total available models. The exact nature of the errors likely to be encountered will be a function of:

1. the governing equation;
2. the method of solution;
3. the input data.

Unfortunately the errors associated with these techniques may or may not result from mutually exclusive combinations of these three factors. For example some of the models of interest may require the solution of sets of simultaneous non-linear equations. One of the most practical ways of solving these equations is by iterating (using perhaps a Newton-Raphson technique) until changes in the dependent variable which constitute the non-linearities are small. Errors associated with this approach may occur if the solution either fails or is slow to converge (a factor in highly non-linear or discontinuous problems). In this case the magnitude of the error will be a function of either a specified tolerance level or specified iteration limit. Although these statements have been made with regards to "outer" iterative schemes the same observations hold true for the use of iterative matrix solution techniques for linear partial differential equations. The poor selection of acceleration parameter in a successive over-relaxation (SOR) technique may result in a failure to satisfy the desired tolerance in the maximum number of iterations. Slow or difficult convergence may also result from the nature of the problem being solved and the selected discretization scheme (this aspect discussed further in Section 2.2.5).

Failure of a numerical solution to converge may also be caused by instability. The stability of a solution will depend on the governing equation, the solution algorithm and sometimes data dependent

(e.g., discretization) factors. As an example consider various finite difference approximations to partial differential equations. The centred-in-time (CIT) approximation of the groundwater flow equation is known to be unconditionally unstable, the backward-in-space (BIS), centred-in-time (CIT) approximation for the convection-diffusion equation is conditionally stable and the backward-in-space (BIS), backward-in-time (BIT) approximation is unconditionally stable for the convection diffusion equation (Intercomp, 1976).

Additional logic errors may occur because the individual running the model may have insufficient knowledge to understand the limitations or restrictions of the model or, in particular, the algorithms employed in the model. The ability of finite element or finite difference solutions to a flow equation to accurately depict the piezometric surface will depend on the nature of the problem and the discretization employed to solve it. An ill-conceived or poorly posed problem, one which is overspecified or physically unrealistic, may also lead to erroneous results.

2.2.5 Data Input

There exists a wide range of errors that could be classified as data input errors. First, and probably most obvious, are those errors where input data does not correspond, either in type or attribute, to the format of the read statement. Even if the type and attribute of the input variable matches that of the required variable this is no guarantee that the value is within a credible or feasible (from a computational standpoint) range. The lack of suitable argument tests within the source code may result in such errors as division by zero, arithmetic indefinites or exponential over(under) flow.

A large percentage of the waste disposal models are numerical in nature and require some form of discretization either in space or time to obtain a solution. The truncation error associated with finite

difference algorithms can be evaluated by examining the magnitude of those terms of the series expansion not considered by the finite difference approximation. Centred-in-time (CIT), centred-in-space (CIS) approximations are second order correct whereas a fully implicit upstream weighting is only first order correct. Finite element techniques typically use the same time discretization schemes as finite difference techniques so the previous comments also apply to the time truncation of these methods. The magnitude of the spatial truncation error is more difficult to evalute owing to irregular element geometries and higher order basis functions. For a triangular element grid using linear basis functions it can be shown that the resulting system of equations reduces to the same as those of a CIT CIS finite difference approximation.

In solving the partial differential transport equations (e.g., solute or heat) it can be shown that the convective term gives rise to a truncation error which has the same appearance as the physical dispersion (Lantz, 1971). This quantity, known as numerical dispersion, is a function of the velocity and some combination of time or space discretization depending on the difference form (e.g., forward, backward, centred) employed. Table 2.1 summarizes numerical dispersion and stability criteria for finite difference approximations to the transport equations (after Reeves, et al., 1985).

2.3 EVALUATION CRITERIA

The criteria developed to evaluate verification methods were selected based on aspects that should be ideally considered by a verification method selected for application to a range of waste disposal models. The criteria were not influenced by limitations that may exist in currently available techniques, but represent the properties of an ideal verification method. The selected criteria include:

Table 2.1 Numerical Criteria for Brine, Heat, and Radionuclide Transport[1,2] (after Reeves et al., 1985)

Scheme	Numerical Dispersion	Dispersion Criterion	Stability Criterion
CIT-CIS	None	None	$v\Delta t/\Delta x+2D\Delta t/\Delta x^2 \leq 2$
CIT-BIS	$v\Delta x/2$	$v\Delta x/2 << D$	$v\Delta t/\Delta x+2D\Delta t/\Delta x^2 \leq 2$
BIT-CIS	$v^2\Delta t/2$	$v^2\Delta t/2 << D$	$v\Delta x/2 \leq D$
BIT-BIS	$v\Delta x/2+V^2\Delta t/2$	$v\Delta x/2+v^2\Delta t/2 << D$	None

1 Here CIT means centered in time, CIS means central in space, BIT refers to backward in time and BIS refers to backward in space.

2 Definition of terms:

V = retarded interstitial velocity
t = time step
x = spatial discretization
D = hydrodynamic dispersion or thermal diffusivity

1. Ability to detect all types of errors;
2. Applicable to all waste disposal models;
3. Machine independence;
4. Ease of use;
5. Identification of percentage of program tested;
6. Cost efficiency;
7. Reliability.

The types of errors that may exist in a computer model application were discussed in detail in Section 2.2 and are summarized in Table 2.2. Appendix A presents specific examples of language errors. Ideally the verification method should be able to detect errors arising from all the possible sources, e.g., language, hardware, data input, numerical algorithm and logic errors. While it is recognized that a single verification method is unlikely to fully satisfy the first criteria, its inclusion is warranted because it will serve to identify those errors that will be (or are likely to be) identified by a specific method/technique.

The second criteria, applicable to all waste disposal models, is meant to identify any limitations of the verification method/technique that might restrict its application to certain types of codes. For example techniques may be language specific.

The third criteria, machine independence was established for the purpose of identifying any hardware specifics (e.g., word size, memory or I/O requirements) that might restrict the universality of the method/technique. This is useful both from standpoint of knowing whether the method/technique is suitable for installation on AECB's current computer, but also for identifying any ramifications that may be associated with upgrading of these resources.

The method/technique should also be evaluated in terms of the degree of difficulty associated with its use. For example the method may require intimate knowledge of the program being tested or knowledge of certain machine languages.

Table 2.2 Types of Errors Found in Computer Models

Classification	Example
Language	data declaration data reference interface comparison I/O errors
Hardware	truncation memory overwrites
Logic	control flow
Numerical Algorithm Errors	unstable or conditionally stable algorithms convergence
Data Input	overflow, underflow numerical dispersion format errors

The verification method should also provide an indication of both the portion of the code that has been accessed by the method or technique as well as the percentage of the code combinations that have been exercised.

Ideally the verification method should also be inexpensive to acquire and operate. Criteria #6 is meant to address this aspect.

The last criteria identified above is the reliability of the method. For example has the method/technique been subjected to some form of QA to ensure that the claims of error detection made by the developer are legitimate. In some cases this may involve using the technique on itself.

The criteria described above will be used to assess the applicability of various verification methods/techniques to waste disposal models. They will not be used to provide a quantitative total rank of individual methods nor is it intended to provide weighting factors which reflect the relative merits of one category over another. The criteria were not selected based on other considerations, such as ease of implementing into a Quality Assurance program or associated requirements because AECB currently has no such requirements in this area.

2.4 VERIFICATION TECHNIQUES AND TOOLS

Verification techniques may be classified as either static, i.e., those that do not require running of the program, or dynamic, i.e., those that do. Static techniques may be further categorized as manual or automated whereas dynamic techniques are sometimes referred to as functional (i.e., those designed to test a specific program function) or structural (i.e., those techniques independent of specific program functions). The following sections describe general techniques and present specific examples of each of these categories.

2.4.1 Static

2.4.1.1 Manual. A number of manual static evaluation techniques exist, most of which may be classified as some form of peer review. Walkthroughs and inspections, for example, are most effective when they make use of teams of three or more people. Suggestions of the roles and qualifications of the team members are numerous (e.g., see Myers, 1979; Andriole, 1983; Miller, 1981) but should include the designer and coder of the program, a moderator whose purpose is to preserve the integrity of the review, potential users and other programmers not involved with the development of the program. Walkthroughs and inspections are similar techniques in that they both involve the use of teams and that they are (at least potentially) capable of detecting all types of errors (i.e., software, hardware, logic or data input).

The two techniques differ primarily in their objectives. Inspections are designed solely for the purpose of detecting errors and usually involve the use of a "checklist" containing common or known problem areas. Walkthroughs on the other hand may have objectives other than, or in addition to, error detection, e.g., program education. Walkthroughs involve the manual simulated execution of various classes of input data. Both of these methods may suffer from their inability to detect a majority of the existing errors with typical efficiencies varying from 30 to 70% for the "total" number of errors (Myers, 1979). IBM was one of the first proponents of walkthroughs and inspections (IBM, 1976).

Desk-checking is a one person review of the program which may be similar in nature to either a walkthrough or an inspection. Desk checking, due to the absence of a team, has been judged to be far less effective than either walkthroughs or inspections (Myers, 1979).

A formal program proof is a manual verification technique that involves the mathematical demonstration of consistency between a program and its specifications. At this time the method is largely theoretical and at best can only be used for small programs or

algorithms. The method has the distinct drawback of requiring in depth knowledge of the design and performance specifications. One example of this approach is the method of inductive assertions developed by Floyd (1967). This approach involves testing the solution specification by writing assertions about the program's input and output. In addition, intermediate assertions are developed for all loops within the program and must completely express the transformations that occur at the point of placement of the assertion. Although formal proofs may be used to detect a variety of error types they have certain serious drawbacks. For example, these methods require detailed and rigorous design specifications which in turn lead to the need for large complex lemmas which require intricate proofs which are highly error prone (Andriole, 1983).

Symbolic verification may be either manual or automated. The underlying approach of this technique is the development of expressions which describe the cumulative effect of all logical decisions and calculations performed along a program path. The major disadvantage of this method is that it becomes unwieldy for either large or nested programs.

2.4.1.2 Automated. A wide variety of automated static analysis techniques exist. Representative techniques include cross-reference mappers, interface analysers, code auditors/verifiers, data flow analysers, control structure analysers, and symbolic verifiers. The following sections discuss each of these techniques, their advantages and limitations and present specific examples of each.

A cross-reference mapper is a highly specialized automatic static verification tool designed to detect a specific kind of software error, such as undefined variables, inconsistent use of variable type or attribute, etc. Cross-reference mappers are relatively inexpensive and easy to use and typically are present as options on most compilers.

An interface analyser is a highly specialized verification tool that examines solely the transfer of variables between program modules. That is, errors associated with inconsistencies between the number, type or attribute of variables passed between subroutines or functions by either argument list or common blocks. Interface analysers are both easy and inexpensive to use and are often incorporated with another verification tool (e.g., the PFORT verifier contains both an interface analyser and code auditor). By themselves interface analyzers have the significant limitation of being able to detect only one type of software error.

Code auditors (also referred to as verifiers or software evaluation systems) are a tool, usually a program, that checks whether an existing piece of software adheres to certain prescribed standards and practices. This would include, for example, such items as syntax, use of structured programming concepts, existence of comments and various conventions dealing with statement labelling and ordering. As such code auditors are not a verification tool in the strictest sense of the word. Auditors may be constructed to detect a variety of software errors and some logic and data input errors. They are language dependent but verifiers for a variety of languages have been developed. Some of the most notable are the PFORT verifier (Ryder and Hall, 1975) developed for PFORT a portable subset of ANSI FORTRAN 66, PBASIC a verifier for BASIC (Hopkins, 1980) and FACES, a FORTRAN verifier developed at the University of California at Berkley. Code auditors have several promising features, they are easy and inexpensive to use (Andriole, 1983). Several large developers of software, such as Bell Laboratories and TRW (Fischer, 1974) make extensive use of code auditors.

Data flow analysers are automatic static analysis tools that were developed to detect errors resulting from variable usage along program paths. Data flow analysers are language specific, a FORTRAN example is the DAVE system of Osterweil and Fosdick, 1976. This particular data flow analyser is representative of the state-of-the-art with respect to these techniques. Its major drawbacks are

associated with: 1) the specialized nature of the software/logic errors detected; 2) fictious errors associated with non-executable paths; 3) inability to handle individual array elements. In addition the technique can be expensive to use although reportedly is relatively easy to learn (Andriole, 1983). At this point in time the approach should still be regarded as experimental in nature.

Automatic structure analysers are useful in detecting a variety of software and logic errors, such as improper loop nestings, unreferenced labels, "dead" code and statements with no successors (Ramamoorthy and Ho, 1975). They can also be used to identify control branches and paths. Structure analysers may also be used to examine the logic flow of the program and in this context cannot be thought of as strictly a verification tool. Control structure analysers require that standards be specified for the program flow. As a result the errors that are detected and the reliability of the method overall will be a function of these standards. Andriole (1983) reports that although the method does not require extensive knowledge or training to use, it is only useful in verifying coarse program properties.

Symbolic verification may be either manual or automated and is sometimes referred to as a dynamic method although it does not involve execution of the program in the normal sense of the word. As discussed in the previous section, symbolic verification involves the generation of output expressions that are functions of input variables and computation path. A number of symbolic verifiers have been developed including DISSECT (Howden, 1977) and ATTEST for FORTRAN language programs. Significant problems exist with symbolic verification, however, and these automated methods should be regarded as experimental. The difficulties are associated with the handling of such items as loops, subscripts, pointers and branching statements. It is generally not feasible to use symbolic verification for all possible paths and consequently the number of errors will be a function of those

paths selected for testing. Symbolic verification shows significant promise in the detection of many kinds of software and logic errors, although it would be less efficient at detecting input and hardware errors.

2.4.2 Dynamic

2.4.2.1 Types of Dynamic Testing. Dynamic testing, unlike static testing, requires the execution of the source code. A number of researchers (e.g., Howden, 1978; Fairley, 1981) have classified dynamic test techniques as either functional or structural.

Functional tests involve the testing of specific program functions and are sometimes referred to as "black box" testing because the person performing the test does not require any knowledge about the internal workings of the code. Any testing performed to check design specifications could be classified as functional tests. There are two major activities associated with functional tests, including the identification of program functions (e.g., branching or computational) and the selection execution and verification of test cases.

Structural testing is designed with the objective of trying to test specific segments of the program. The segments may be branches, paths or specific statements of the program. Structural testing is sometimes referred as "white box" testing (Myers, 1979) because the person performing the test requires some knowledge of the internal workings of the program.

Various types of testing may be classified as either functional or structural. Exhaustive testing is one such example. If test cases are designed to execute all possible loops within a program then the method should be classified, in the strictest sense of the word, as a structural test. If exhaustive test cases are constructed based on the premise of executing all feasible combinations of input

variables then the approach could be classified as a functional technique. The distinction between the two may be a moot point for this particular example. Exhaustive testing is typically impractical for all but the smallest programs.

Assertion testing is another example of a test technique that may be either structural or functional. Assertion testing involves the generation and processing of assertions. Assertions are internal checks placed within the code that are used to verify internal operation of the program. Assertion testing may be considered as functional if its objective is to check intermediate values to determine if they are either in range or computed correctly. If the assertion is a check on a control variable for the purpose of establishing whether a path has been accessed the approach can be classified as structural.

A number of miscellaneous dynamic based tools also exist including interactive test aids, compiler based tools and truncation analyses. Interactive test aids (e.g., debuggers) are primarily used for error correction rather than error detection. They can be used to verify memory storage, intermediate calculations and program flow. To be used effectively, interactive test aids should be used in conjunction with a predetermined, organized, testing procedure. Interactive test aids can be applied to most types of source code, however, most currently available tools are language specific. Many large developers of software, such as NASA (Taylor et al., 1979) and the Sperry Rand Corp. (Sperry Rand, 1979) make use of such interactive aids in testing their software.

Compiler based tools consist of a set of optional tests which may be conducted during execution arising from conditions set during compilation. The tests are designed to examine such aspects as array index checking, uninitialized value checks, nil pointer checking, and errors arising from mathematical operations. Drawbacks with this method are associated primarily with execution speed/code size tradeoffs. Tools associated with compilers are obviously language specific.

A number of researchers (e.g., see Miller, 1975; Battye and Parsaei, 1980) have developed approaches to quantifying the roundoff that occurs for software coding of algebraic processes. While such approaches have demonstrated some merit they have some severe limitations associated with them (e.g., only specific algorithms, inability to handle loops/branches) and at this point in time should only be regarded as experimental.

The verification method that is most widely used for waste disposal models is comparison based testing. Testing is performed by analyzing the same problem using two different approaches (one being the program to be verified) and comparing the results. The second set of results may be obtained by use of another (hopefully already verified) program or possibly by evaluating an analytical solution. Difficulties associated with this method include:

1. Selection of input data that will adequately test all aspects of the program;
2. Ability to find a suitable analytical solution or alternate program for comparison
3. Resolving differences between two unverified programs (i.e., which one, if any, is correct?)

Multiple code comparisons, such as INTRACOIN and HYDROCOIN (Andersson, 1985; INTRACOIN, 1984; Jansen and Hewitt, 1986), are one approach towards resolving items 2 and 3. Such code comparisons also point to the need for sets of carefully planned test cases to adequately test various aspects of a model.

2.4.2.2 Difficulties of Dynamic Testing. Regardless of the classification of the test, all dynamic testing involves the generation of input data, execution of the program and verification of the results. Input data may be generated using a number of approaches (Howden, 1978; Myers, 1979) including boundary value or extremal testing and special (e.g., error guessing) testing. Some data

generating approaches have been automated to generate the data in a more systematic deterministic manner (Howden, 1975). Regardless of the method or philosphy used to generate input test data, a number of problems are associated with dynamic testing techniques:

1. Ascertaining the correctness of the output results;
2. Estimating the effectiveness of the test.

The first of these, the ability of ascertain whether the output is correct, is a major difficulty especially in the area of waste disposal models. Unless the program either fails to execute (i.e., due to an execution error) or produces obviously erroneous results (e.g., negative concentrations), it may be extremely difficult to establish the correctness of the program output. For numerical models analytical solutions may, under certain circumstances, be used to confirm the correctness of the output. For multi-dimensional coupled process applications (the kind likely to be encountered in a waste disposal program) however, few analytical solutions exist to verify the output and generally it is assumed that the multi-dimensional aspects are coded correctly if a one or two-dimensional solution can be proven correct.

The second major problem area associated with dynamic tests is the ability to assess the effectiveness of the generated input data sets in "trapping" program errors. One approach to this problem is mutation analyses or error seeding (Budd, 1981). In this approach, a number of errors are artificially placed within the program. The number of these artificial errors discovered by the input data sets will provide (ideally) an estimate of the effectiveness of the specific set of test data. Unfortunately, one of the major difficulties associated with such an approach is the manner for selecting the errors to be introduced. The questions of whether the "seeded" errors are biased or are representative of the errors that exist naturally in the program are difficult to answer and, at this point in time, have not been fully addressed.

An other approach used to evaluate the effectiveness of test data sets are coverage metrics (Stucki, 1973). This approach involves the use of a measure to estimate the percentage of the statements accessed by the test data. Unfortunately, the difficulty associated with establishing the correctness of the output remains the major limitation of these methods. Still, the method is useful in identifying the fraction of the code that has been totally untested.

The automated generation of test data also has problems associated with it. Andriole (1983) reports that the benefits obtained through the use of such techniques rarely justifies the significant costs associated with their development. He goes on to say that such tools are seldom used in actual test environments and that they are among the most expensive testing tools that exist today.

2.4.3 Overview of Verification Techniques

Table 2.3 presents a summary of representative generic testing techniques and their ability to satisfy the criteria developed in Section 2.3. It became apparent from our review that most techniques are very specialized, designed to test specific aspects of the code. This is especially true of automated techniques, such as cross-reference mappers, interface analyzers, data flow analyzers and control structure analyzers. Manual static techniques, such as walkthroughs and inspections are (or at least can be) more general in terms of the types of errors that can be uncovered.

It also became apparent that many techniques (some of which showed significant promise) were still experimental in nature and some could only be described as being theoretical, having little practical use. This would apply, for example, to such verification approaches as formal proofs or symbolic verification.

Table 2.3 Summary of Verification Tool/Technique Evaluation

Test Technique/ Tool	Type[1]	Example	Errors Detected	Program Restrictions	Machine Restrictions	Ease of Use	Test Estimate	Cost[2]	Reliability	Comments
Walkthrough	M,S	-	All	none	none	easy	no	$	variable	effectiveness a function of team
Inspection	M,S	-	All	none	none	easy	no	$	variable	effectiveness a function of checklist
Deskchecking	M,S	-	All	none	none	easy	no	$	poor	
Proof of Correctness	M,S	-	All	none	none	difficult	no	$$ to $$$	moderate	
Cross-Reference Mappers	S,A	Compiler Options	Language - undefined variables - mismatched type, attribute	language specific	not specified	easy	no	$	good	very specialized for one error type
Data Flow Analyzers	S,A	DAVE (Osterweil and Fosdick, 1976)	Language - data flow type	language specific	none	easy	no	$$$	moderate	only detects specific error types
Interface Analyzers	S,A	PSL/PSA	Only Language Interface	language specific	not specified	easy	no	$	moderate	
Control Structure Analyzer	S,A	Howden, 1976	Language, Logic - program control	none	not specified	moderate	no	$	moderate	can only verify coarse program properties
Code Auditor/ Verifier	S,A	PFORT (Ryder and Hall, 1975)	Language Logic	language specific	not specified	easy	no	$	good	
Symbolic Verifier	S,M,A	DISSECT (Howden, 1977)	Language Logic	language specific - simple codes only	not specified	moderate to difficult	no	$ to $$$	moderate	unwieldy for large programs or loops. theoretical tool only

Table 2.3 Summary of Verification Tool/Technique Evaluation (cont'd)

Test Technique/ Tool	Type[1]	Example	Errors Detected	Program Restrictions	Machine Restrictions	Ease of Use	Test Estimate	Cost[2]	Reliability	Comments
Assertion Testing	D	Stucki & Fushee, 1975	Logic Language	no	no	moderate	no	$$ to $$$	variable	few automatic tools available
Interactive Test Aids	D	debuggers	Hardware e.g.,memory allocation Logic Software	language specific	no	easy	no	$$	variable	only effective as part of an overall test strategy
Compiler Based Tools	D,A	compiler option	Language - array index - initialization - nil pointer - math errors	language specific		easy	no	$ to $$$	good	costs involve execution time/code size tradeoff
Roundoff Analyses	D,M	Miller, 1975	Hardware	algorithm specific	no	moderate to difficult	no	variable	moderate	very specialized. does not address all aspects of numerical solution
Input Data Generators	D,A M	Panzi, 1978	Language Logic	some are language specific		easy	program coverage	$$$	poor	difficult to ascertain correctness of output
Comparison	D	- to analytical solutions - code comparisons	Language Logic	no	no	easy	no	$ to $$$	variable	reliability a function of test cases

1 S = static
D = dynamic
A = automated
M = manual

2 $ = inexpensive
$$ = moderate cost
$$$ = expensive

Dynamic testing techniques have the major difficulty of establishing whether the output is correct. This fact limits the usefulness of automatic input data generators for waste disposal models. Roundoff analysis tends to be too highly specialized to be of practical use. The fact that assertion testing requires significant training in order to generate meaningful assertions and the lack of automated processors renders this approach less attractive than others. Interactive test aids (such as debuggers) can provide valuable assistance in performing verification tests but cannot be viewed as stand-alone technology. They should be used as a component in an overall test strategy. Compiler based tools, although sometimes very specialized are worthwhile for consideration because they can typically be included at marginal cost. Comparison based testing is perhaps one of the most appropriate for waste disposal models. The comparison format overcomes the difficulties of verifying the correctness of the output. The major difficulty associated with the approach is the selection of suitable test cases that will exercise all important program functions.

2.5 VERIFICATION OF WASTE DISPOSAL MODELS - STATE-OF-THE-ART

As part of this study, INTERA undertook a survey of agencies and organizations that use and/or develop waste disposal models. The agencies were requested to provide information describing any official verification and validation requirements or standards to which they were subjected and the approaches or procedures adopted to meet these standards. Table 2.4 lists the agencies that were contacted and summarizes their response.

In general, it was found that although most of the respondents recognized the need for a formal verification and validation plan that could be applied to waste disposal models only a handful had actually taken steps to implement such a program. Of these

Table 2.4 Status of Verification/Validation Plans of Agencies using/developing Waste Disposal Models

Organization	Description
AECL (Atomic Energy of Canada Ltd.)	- draft documentation unavailable at this time - QA program developed specifically for SYVAC - verification consists primarily of inspection, comparison - validation to be resolved
BWIP (PNL) (Basalt Waste Isolation Program - Pacific Northwest Labs)	- draft documentation submitted to USDOE (Richland) - USDOE unwilling to release
EPA (U.S. Environmental Protection Agency)	- no formal verification/validation requirements - responsibilities lies with individual project managers
LBL (Lawrence Berkeley Labs)	- no formal verification/validation requirements
LLL (Lawrence Livermore Labs)	- no formal verification/validation requirements
LAL (Los Alamos Labs)	- no formal verification/validation requirements
NUTP (National Uranium Tailings Program)	- no formal requirement - all actions taken exclusively for UTAP - includes peer review and code comparison - no validation planned; only application intended at this time
NRC (U.S. Nuclear Regulatory Commission)	- verification/validation requirements summarized in Regulatory Guide 1.152
ORNL (Oak Ridge National Lab)	- draft program in place to satisfy NQA-1 - consists of review panel and hand calculations - unverified portions must be identified

Table 2.4 Status of Verification/Validation Plans of Agencies using/developing Waste Disposal Models (cont'd)

Organization	Description
OWTD (formerly OCRD) (Office of Waste Technology Development)	- draft plan in place - addresses most of NQA-1 requirements - documentation unavailable
Salt site assessment - INTERA	- program in place - verification consists of internal and external peer review and comparison based testing
WIPP - Sandia (Waste Isolation Pilot Plant)	- contractor furnished software subject to EP401422 which provides for execution of test cases - formal QA plan to satisfy NRC requirements proposed in NUREG/CR-4369
SRL (Savannah River Labs)	- no formal requirements
SKI (Swedish Nuclear Power Inspectorate)	- no information received

the majority of the verification and validation protocols were part of a draft or preliminary Quality Assurance (QA) Plan. Several of the respondents (e.g., AECL, Pacific Northwest Laboratories, Office of Waste Technology Development (formerly OCRD)) were unwilling to release any such preliminary information. Some draft information was obtained from Oak Ridge National Laboratories. From the information received and INTERA's familiarity with waste disposal models, it would appear that the vast majority of computer models developed in this area have been subjected to little formal verification or validation. In the United States the responsibility for this area lies with the United States Department of Energy (DOE) or its contractors as part of its compliance with the United States Nuclear Regulatory Commission's high level waste regulation 10 CFR 60 or the United States Environmental Protection Agency Standard 40 CFR 191. Neither of these rules however, address verification or validation requirements specifically, and as a result the onus would appear to be placed on DOE and its subcontractors to demonstrate the "reasonable assurances" required by such rules.

For this reason DOE has selected various standards and regulations as the basis of its requirements for its contractors. The standards would appear to be based on those recommended under NRC's Regulation 1.152 which provides criteria for programmable digital computer system software systems of nuclear power plants. The adopted standards and regulations vary slightly from program to program but generally include:

- Title 10 Code of Federal Regulations, Part 50, Appendix B, "Quality Assurance Criteria for Nuclear Power Plants and Fuel Reprocessing Plants", January, 1977.

- ANSI/ASME NQA-1, "Quality Assurance Program Requirements for Nuclear Facilities", 1986.

- NUREG-0856 (ANSI Standard No. 413)," Final Technical Position on Documentation of Computer Codes for High Level Waste Management", June, 1983.

In addition, program specific standards include:

- Quality Assurance Specification for Oak Ridge National Laboratory, September, 1986.

- Engineering Procedure, Quality Program requirements for contractor furnished software, Sandia National Laboratories, September, 1983.

- ONWI QA Spec. E512-11600 (including attachement B).

The requirements imposed by these standards in the area of model verification are typically very general and involve providing a definition of the process and a statement of the requirement. ANSI/ASME NQA-1, for example, defines verification as "the art of reviewing, inspecting, testing, checking, auditing or otherwise determining or documenting whether items, processes, services or documents conform to specified requirements". The application of computer programs is covered by Section 3 of the standard which states that individual verification is not required provided (in part) "the computer program has been verified to show that it produces correct solutions for the encoded mathematical model within defined limits for each parameter employed".

10 CFR 50 Appendix B is equally as general in providing quality assurance criteria for nuclear power plants. This regulation does not address either verification or validation of computer software directly but does provide overall quality assurance criteria for nuclear power and reprocessing plant "structures, systems and

components". It goes on to state that one of the quality assurance functions is "verifying, such as by checking, auditing, and inspection, that activities affecting the safety related functions have been correctly performed". The regulation does not specify however either how this is to be accomplished or what constitutes adequate verification.

NUREG 0856 summarizes NRC's position on the documentation of computer models. As such, it is not a regulation per se but merely a suggested guideline. It recommends the creation of a document describing model assessment and support. This should include a description of both tests performed by the code developer and independent assessment and should include input decks, descriptions, and results for specific tests. The aspect of the model addressed by each test and the models performance should also be discussed.

The Salt Repository Project Office has placed the following software verification requirements on its contractors:

- The contractor shall test and verify computer software by:

  1. Comparison of results with those of other verified software;
  2. Comparison of results with those from analytical solutions.

- Verification procedures shall consist of a test plan identifying the test cases, input data and expected results;

- New software shall be tested for those input and conditions necessary to fully exercise the software and identify acceptable boundaries.

The Salt Repository Office does not provide any more detail as to how these requirements are to be satisfied.

Sandia National Laboratoires places a similar set of general requirements upon its contractors as those of NQA-1. The requirements define verification as "a process that demonstrates the consistency, completeness and correctness of the software". The requirement goes on to specify that the contractor shall provide for verification, including activities, such as:

- inputting of both normal and abnormal data and exercising the software under expected use conditions;
- testing to identify the operating boundaries of the software and establish a benchmark for future comparisons;
- documenting the results of the verification exercises;
- documenting and storing verification procedures and test cases for future use in verification of revised versions of the software."

The above paragraphs describe the requirements imposed by the United States Department of Energy on its contractors. The following section describes plans that were developed to meet these requirements. Generally, the verification and validation aspects are part of a larger quality assurance plan.

The draft quality assurance plan of Oak Ridge National Laboratories provides procedural instructions for verification of computer codes. The program identifies three approaches to verifying a computer code:

1. Comparison to hand calculations;
2. Comparison to an analytical solution or approximation;
3. Comparison to a verified code designed to perform the same type of analysis.

The verification procedure involves the development of a code specific plan. The plan must identify the computer code, the approach to be followed and the documentation that will be required. The plan should also describe the mathematical models and numerical methods of the code. In a very complex code, the verification may proceed on a piece by piece basis. The verification plan shall be submitted to the responsible manager for review and approval. The program recommends that either the staff member performing the verification or the reviewer should be independent of the code development effort.

The results of the verification must be summarized in a report which describes:

1. The computer code being verified (including name and version);
2. Verification method;
3. Verification results including how the code must be modified if the verification were unsuccessful or, if the verification was successful.
   i) the conditions (i.e., input ranges) for which the code has been verified,
   ii) conditions which remain unverified (code segments, run options and ranges of input variables).

Sandia National Laboratories (SNL) was responsible for the development of a Quality Assurance Program for computer software created at SNL for the NRC. It is anticipated that the NRC will make use of these computer codes when assessing DOE license applications. The program developed at SNL for NRC specifies the verification

requirement as including "adequate tests to demonstrate that the program satisfactorily performs its stated capabilities". The QA plan goes on to acknowledge that because each program is different, an exact definition of program testing requirements cannot be specified and, as a result, it places the responsibility of whether a code has been adequately tested onto the principal investigator.

The Salt Site Performance Assessment Project Quality Assurance Program was developed by INTERA Technologies Inc. for the DOE's Office of Nuclear Waste Isolation (ONWI). The plan addresses the verification of computer software by requiring that results are verified for consistency with:

1. problem assumptions;
2. numerical techniques or algorithms selected for solution;
3. defined and specified boundaries, range limits for variables, parameters and constants employed.

The Salt Site Performance Assessment QA plan is also specified in terms of the relevant sections (and levels) of INTERA's own QA program. With regards to software verification, this includes defining the scope of the procedure, providing definitions of the variables, outlining the responsibilities of the personnel involved, the procedure to be followed and the acceptance criteria. Data to be used in the verification process should:

1. be realistic for the intended model application;
2. be the best practically available;
3. be fully documented as to source, method of acquisition, accuracy, and reliability.

The program also outlines the documentation and archival requirements of the verification test.

Based on our review of formal verification and validation requirements and plans of agencies and organizations using and/or developing waste disposal models, this area has only recently received any significant attention. In most cases, Quality Assurance programs that are meant to address these concerns are just now being formulated. Based on the draft information that was reviewed, the standards/ requirements and the plans/approaches are written in very general terms. While key items are sometimes identified in the standards, the specific approaches to be followed and the criteria to judge the verification effort are not specified. This short-coming may not be industry specific. Software development standards for the U.S. military are equally as vague with regards to ascertaining whether formal tests adequately satisfy design requirements (U.S.D.O.D., 1985).

3. VALIDATION

3.1 DEFINITION

Validation, like verification, is a word which has no universally accepted definition. Quite often validation is used interchangeably with verification. For the purpose of this study validation has been defined as the process by which one becomes confident that the predictions of the model describe the real behaviour it is asserted to represent. Boehm (1984) uses the following questions to differentiate between the two processes:

Verification - am I building the product (model) right?
Validation - am I building the right product (model)?

It is important to recognize that this definition of validation implies that it is a function of the application. Consequently, a model which is valid for one application may be invalid (or totally inappropriate) for another. Validation forms an essential element of the overall program benchmarking process, as described in Section 2.1.

3.2 COMPONENTS OF AN IDEAL VALIDATION

In general, the only accepted validation approach is comparison to data or observations. This comparison must be done without adjusting the input data or model parameters to achieve a better fit, which would reduce the validation to a calibration exercise. An ideal validation would correspond to a system where the calculated output could be compared to measured data corresponding to the same conditions. Unfortunately, for waste disposal models, validations are typically much less than ideal.

It must be recognized that the validity of a model to represent a specific process is a function of both the model and its application. A model cannot be validated apriori for all applications and considerable onus must be placed on the person using the model to ensure that model and its application are compatible. Under ideal conditions, the following would have to be demonstrated to ensure that the model is validated:

1. Demonstrate that the processes and mechanisms relevant to the application are described adequately by the model. Ideally each and every process/mechanism should be validated both independently and in conjunction with other processes.

2. Demonstrate that the assumptions inherent in the model will not be violated by the intended application.

3. Define the range of the input variables over which the model can be estimated to perform reliably.

4. Estimate the reliability and/or bias of the answers provided by model (quantitatively) and how this reliability might change as a function of input variables or violation of assumptions.

It should be emphasized that the requirements outlined above are idealistic and may not be feasible or practical in reality. To adequately fulfill the first requirement for a groundwater transport model which is being applied to potential repository would ideally require application of the model to the site for the purpose of validating the transport processes over the time and spatial scales relevant to the end objective. This is clearly infeasible given that the time and spatial scales of interest are on the order of 1000 to 10,000's years and 10's to 100's of kilometers.

The second and third requirements identified above recognizes the fact that many of the models are theoretically based and make use of ideal behaviour or conditions or that the laws governing the relationship are valid over a finite range. In fluid flow problems, for example, different relationships are used to describe energy losses depending on the Reynolds Number of the system. In the solid mechanics field, the same can be said for elastic versus plastic deformation problems.

The fourth requirement is intended to provide an indication as to the overall ability of the model to represent real physical processes. The reliability of the model will be determined by the error that exists between model output and observed data. This error will be due to a number of factors, such as the accuracy of the data (e.g., measurement error) and the validity of assumptions and approximations.

3.3 DIFFICULTIES ASSOCIATED WITH VALIDATION OF WASTE DISPOSAL MODELS

There are a number of problems that exist that are specific to the validation of waste disposal models. Some of these problems are associated with the fact that validation is application specific. Other difficulties are associated with the fact that the processes governing the migration of nuclear waste are still poorly understood and often the models constructed to represent these processes do not yield unique solutions. Both of these aspects are discussed in detail in the following sections.

3.3.1 Application Difficulties

Part of the difficulty associated with waste disposal model validation is that a number of different processes contribute to the total contaminant migration process. Under one set of conditions (or application) a specific set of processes may govern waste migration while for another a totally different set may dominate. Consider for example a shallow low level waste facility. If the facility is located in a fairly permeable medium (e.g., sand) then convection and dispersion will dominate the transport process. If the facility is located in lower permeability material (e.g., clay) then diffusion may well control the rate of contaminant transport.

In addition to the question of whether the processes controlling the migration are valid, the question of whether they are representative over the scale of the application must also be addressed. It has been theorized, for example, that certain waste transport parameters, such as dispersivity may be scale dependent (Pickens and Grisak, 1981; Schwartz, 1977). Although the incorporation of scale dependent parameters into a solution of the transport equation may be straightforward, the major difficulty lies in acquiring sufficient data to allow the formulation of a correct relationship.

The same difficulties of scale also pertain to the time domain as well as that of space. In fractured porous media, for example, it is now believed that matrix diffusion will represent a significant sink for migrating dissolved solute. Although relationships describing this phenomena have been formulated and compared to field data (Moreno et al., 1983) it remains to be seen whether such relations would be representative of long term matrix diffusion. The vast majority of tests investigating matrix diffusion effects have been conducted over relatively short time scales and the fitted models basically assume an infinite rock matrix. Whether such an assumption would be valid over thousands or tens of thousands of

years has not been demonstrated. Another difficulty associated with the relatively short term data available to describe matrix diffusion effects is the amount of potential "noise" in the data. Short term diffusion tests only utilize the rock matrix in the vicinity of the fracture surfaces. This rock is more likely to be influenced by the fracture (e.g., density of micro-cracks) than the bulk rock matrix. In addition such phenomena as surface adsorption or ion-exchange may play a larger role at the fracture interface and their effects may be mis-interpreted as matrix diffusion.

The examples cited above highlight another validation difficulty associated with model application - the ability to collect sufficient data to characterize the site. Even once the processes governing waste transport are known it may be difficult to collect sufficient data to adequately characterize the site. The heterogeneities that exist in natural geologic systems and the scale over which they are required to be characterized may render definition of the hydrogeologic regime difficult. In addition, even if a site were completely defined through field investigations, existing computer hardware restrictions may limit the ability to incorporate this data into a model.

To some extent therefore the question as to whether a model has been suitably validated must be answered by the individual using the model or reviewing its application. He or she must decide if the validation tests conducted to date are satisfactory for the application at hand. This would involve checking such items as dominant processes, time and space scales, range of input parameters and adequacy of data base.

3.3.2 Uniqueness

The forgoing discussion highlighted the fact that for some waste transport problems the processes controlling waste migration are poorly understood. This has contributed to the number of poor quality validations reported in the literature. Some researchers have constructed models that have new mechanisms added into them in order to better fit the observed data. Often, however, such mechanisms have little scientific basis and in these instances the model application becomes little more than a curve fitting exercise. Moreno et al. (1983) demonstrated the ability of four different models, including four different combinations of mechanisms, to replicate the breakthrough concentration of strontium and iodine tracer tests. The question of which model (if any) adequately represents the physical system remains unresolved.

The experiments of Moreno et al. (1983) indicate the existence of another problem area in model validation, that of uniqueness. Too often "validation" of models consists of a single application and comparison to field data. The concept of uniqueness is typically not addressesd. As Moreno et al. (1983) demonstrated it is easy to obtain multiple parameter fits to single data sets. Only once the number of degrees of freedom can be established can a sufficient number of tests be designed to ensure model validation. Barring this approach it should be recognized that confidence in model validation increases with the number of tests fit by the model. Increased confidence is alsc associated with a model that has been used to predict a phenomena as opposed to post-experiment curve fitting or calibration.

3.4 VALIDATION OF WASTE DISPOSAL MODELS - STATE-OF-THE-ART

As discussed earlier (Section 2.5), a survey was conducted to determine the verification and validation requirements of agencies and organizations using and/or developing waste disposal models and the approaches/plans adopted to meet these requirements. The standards/regulations that dictate the validation requirments are generally the same as those that govern the verification requirements. Briefly these include:

- 10 CFR 50, Appendix B, "Quality Assurance Criteria for Nuclear Power Plants and Fuel Reprocessing Plants", January, 1977.

- ANSI/ASME NQA-1, "Quality Assurance Program Requirements for Nuclear Facilities", 1986.

- NUREG-0856 (ANSI Standard No. 413)," Final Technical Position on Documentation of Computer Codes for High Level Waste Management", June, 1983.

Organization specific requirements include:

- Quality Assurance Specification for Oak Ridge National Laboratory, September, 1986.

- Engineering Procedure, Quality Program requirements for contractor furnished software, Sandia National Laboratories, September, 1983.

- ONWI QA Spec. E512-11600 (including attachement B).

As with verification, the requirements imposed by these standards in the area of model validation are very general. For example, ANSI/ASME NQA-1 states that a condition for computer programs being used for design analysis is that "the encoded mathematical model has been shown to produce a valid solution to the physical problem associated with the particular application".

NUREG-0856 provides general guidelines on how descriptions of code validation should be included in its documentations, e.g., "Describe the program for evaluation and validation of the code. Include past, ongoing, and planned future activities. Give descriptions, input files and results for specific tests".

The Salt Repository Project Office has placed the following software validation requirements on its contractors:

- Validation must consist of demonstrating the adequacy of the software and resultant data by comparing software results against verified and traceable data obtained from laboratory experiments, field experiments or observations or in situ testing. Data used in the development process is precluded from being used in the validation process.

- If data from the sources identified above is unavailable alternate approaches (e.g., peer review, comparisons with the results of similar analysis) shall be documented including an evaluation of the degree of validity of the model.

Sandia National Laboratories has the following specific requirements in the area of software validation (Sandia, 1983):

- The contractor shall provide for validation of the software in actual use in processing data supplied by or developed for Sandia under contract. Validation includes such activities as:

  1. Inputting of typical data representative of the application being validated and exercising the software under expected use conditions;

  2. Correlating the software results with actual data, such as that obained in the field;

  3. Documenting the results of the validation exercises.

The above sections describe typical requirements of agencies developing/using waste disposal models. To address these (and other) requirements Quality Assurance (QA) plans have been developed. The following paragraphs discuss the validation aspects of some representative QA programs.

The draft quality assurance plan of Oak Ridge National Laboratories provides procedural instructions for the validation of computer codes. The procedure involves:

1. Assigning responsibilities;

2. Preparation of a validation plan which:
   a) identifies the computer code (and version) to be validated
   b) describes the approach to be followed in validating the code
   c) describes the documentation which is required
   d) If the validation is partial, identify which code functions or parts will be validated;

3. Submit and receive approval from responsible manager;

4. Establishment of a review panel;

5. Performing calculations in accordance with QA procedures;

6. Documenting in a report:
   a) computer code and version;
   b) method of validation
   c) results and success of validation
   d) conditions (options, ranges of input) for which the model has been validated. Conditions for which the model/code remain unvalidated;

7. Review of the validation by the review panel, resolution of difficulties and approval of validation report;

8. Archiving validation plan and report.

A Quality Assurance (QA) program was developed by Sandia National Laboratories (SNL) to be applied to computer software developed at SNL for the NRC. The quality assurance program makes an interesting comment on the state-of-the-art with respect to validation of waste management models:

"In waste management applications, the object of validation is the assurance that the model embodied in a computer program is a correct representation of the process or system for which it is intended. Validation is generally accomplished by using computer programs to simulate field or laboratory experiments. For cases where these experiments are well defined, validation is possible. However, parameters that control experiments involving groundwater flow and contaminant transport, especially in fractured rock, at this stage are

too poorly defined to allow for the validation of computer programs. In these cases, the exercise of applying the programs to field experiments is still a valuable test but falls short of validation".

As a result the Sandia program developed for NRC does not specify any actions for validating computer software.

The Salt Site Performance Assessment Project QA program addresses the validation of software by specifying:

- software shall be validated through comparison of results against fully verified and traceable data from laboratory experiment, field experiments or observations or in situ testing;

- approaches other than those specified above shall be documented and an estimate of the degree of validity of the model shall be made;

The validation requirements are specified in terms of INTERA's own QA program. Specific aspects identified include defining the scope of the validation procedure, outlining responsibilities of the personnel involved, the procedure to be followed, development of acceptance criteria and the documentation of the validation exercise. The documentation shall include a description of the limitations/restrictions of the model. Archival requirements for the validation procedure are also identified.

The above discussions summarize the formal validation requirements and approaches to the waste disposal model validation area. In addition to these formal approaches, it is useful to examine commonly employed approaches for validating such models. In general, the only accepted and practised validation approach is comparison to

data or observations. The only difference among various validation approaches is the source of this data. The most common of these include:

1. Laboratory Experiments;
2. Field/In situ Experiments;
3. Natural Analogues;
4. Reasoned estimates and bounding analyses.

Laboratory and field experiments both suffer from problems of the representativeness of the spatial and/or temporal scales compared to those associated with the area and times of influence of a waste repository. Another common difficulty associated with experiments and field studies is the lack of quality control. The need for tightly controlled experiments to provide reliable data cannot be overemphasized. In spite of these difficulties, and due to the lack of data at more representative scales, the "validation" of waste disposal models using data from these sources is commonplace.

Another source of data used in validating waste disposal models is natural analogues. National analogues, such as the natural reactor at Oklo or the geothermal anomoly at Marysville, have the advantage of a more representative time-scale than that associated with laboratory or field data. A number of data deficiencies have been identified for most natural analogues (Andersson, 1985) including insufficient data to characterize initial conditions and to define the hydrologic regime over the time period of interest. In spite of these difficulties, it should be recognized that natural analogues can play a useful role in the validation of waste disposal models. Difficulties associated with data deficiencies can, to some extent, be overcome by analyses based on either bounding or most likely parameter values.

Reasoned estimates and bounding analyses are sometimes used as an approach to estimating data that cannot be measured directly. Reasoned estimates involves the use of individuals who are knowledgeable about the dominant processes to estimate the anticipated behaviour. Bounding analyses are calculations that are performed using upper limits on unknown data.

The comparison of an unvalidated program to one that is validated cannot be viewed as a separate validation approach. The second model is bound by all the restrictions/limitations imposed by the data used to validate the first model. Hence this approach is nothing more than validation by the original data.

The data deficiencies identified above point to the need for ways of completing the data base. One such method is through the use of peer review groups or expert opinion. While it is recognized that many experts may disagree on the interpretation of data or forecasting of conditions, it must also be recognized that such opinion may currently represent our best and perhaps only alternative to having the data in hand. Expert opinion, however, should not be viewed as a standalone validation technique but should be used to augment the existing data base and could change or evolve during a validation exercise.

3.5 VALIDATION CRITERIA

Given that it is unlikely that a waste disposal model can ever be considered as fully validated and will continue to remain a function of the application, a set of criteria has been developed to assist the AECB in its evaluation of validations reported in the literature as well as assessing the model's suitability to a specific application. These criteria are meant to be used to evaluate specific validation exercises as opposed to different approaches (due to the

fact only one approach - comparison with data is in use). The criteria can best be presented in terms of a number of questions to be asked by the reviewer:

1. Does the validation identify restrictions/limitations of the data? This would consist of identifying the dominant processes and mechanisms represented by the data and their time dependence (steady state versus transient), scale (temporal and spatial) and dimensionality. The range of input data covered by this validation should be identified or, at least, specific values that would not be covered should be identified. The source of the data (e.g., reasoned estimates, measured values) and the accuracy of the data due to measurement, equipment limitations as well as uncertainties resulting from unknown or assumed data (e.g., homogeneous, isotropic with respect to permeability) should be identified. Large error ranges that could potentially mask calculated trends should be a consideration under criteria for acceptance. This concern points to the need of using data from well controlled experiments. The existence of any empirical coefficients should be identified and the difficulty associated with their evaluation as well as the output sensitivity should be described.

2. Does the validation identify the theoretical basis, the processes it is intended to represent and any restrictions/limitations of the model on the validation exercise? This would require the identification of fundamental assumptions associated with the governing equations, such as constant density and viscosity, isothermal, valid flow regimes, trace contaminant concentrations, equilibrium adsorption, etc. The ability

of the data set to satisfy these assumptions should be demonstrated with any deviation being quantified in terms of its effect on the model output. The validation should also identify restrictions associated with the numerical algorithm or solution technique. For example, restrictions on gridding or time step imposed by Peclet or Courant Numbers should be identified and compliance for the validation at hand should be demonstrated.

3. Does the validation explain and justify its criteria for acceptance? A model that employs a statistically based analysis of the observed versus calculated data may be better able to provide an unbiased evaluation of the success of the validation effort than a visual inspection. The model validator should also evaluate the uniqueness of the calculated set of data points. Is it possible to obtain a comparable fit using radically different values of the input parameters together with the same model. The sensitivity of the fit to the model input parameters should also be investigated. The validity of model parameters should be examined in terms of their physical meaning. This is especially true for variables that are calibrated. Often inexperienced individuals attempting to fit observed data will change input values outside of their known or credible range in order to marginally improve a fit. A classic example of this problem is the estimation of specific storage from curve fitting techniques. In a saturated confined system the storativity is comprised of a fluid and formation compressibility. In the vast majority of cases, the fluid is more compressible than the formation and therefore the fluid compressibility can be used to set an upper bound for this quantity.

The validator should also identify any differences that may exist between the validation and the intended waste disposal application. For example, are the dominant processes, scale, and quantity/quality of data the same? Have differences between the validation and final (prediction) application been examined and quantified in terms of their impact on the results?

The above criteria were developed to assist AECB in evaluating model validations in the context of their appropriateness and suitability for specific applications. Other Quality Assurance aspects affiliated with the validation exercise, such as the development of test specifications, documentation and archiving are beyond the scope of the current report.

4. CONCLUSIONS AND RECOMMENDATIONS

Our review of agencies/organizations using/developing waste disposal models would indicate that the verification and validation of these programs has only recently been recognized as an area where formal requirements are needed. As such, most of the Quality Assurance programs intended to address these requirements are in the development stage. Based on our review of draft documentation, it can be said that the requirements/approaches of the standards/programs reviewed are quite general and do not provide specific guidelines as to what is required (or how to go about) verifying and validating a computer model.

The verification of waste disposal models is confronted with the same types of problems associated with other programs. In general, programs are subject to errors from five sources including language, hardware, logic, numerical algorithm, and data input. A variety of verification tools have been developed to detect program (mostly software related) errors. Frequently, these tools are classified as either static, if they do not require execution of the program, or dynamic if they do. Problems associated with some of the static techniques include:

1. Too theoretical to be of practical use (e.g., formal proofs);
2. Only experimental prototypes available at this stage (e.g., symbolic verifiers);
3. Very specialized with only selected error types being considered (e.g., cross-reference mappers, interface analyzers, etc.).

The major problem associated with dynamic test methods is ascertaining the correctness of the output. Exhaustive testing or bounding analyses are of little utility except for detecting execution errors resulting in program termination or flagrant errors related to

obviously incorrect output (e.g., significant negative concentrations). Of the dynamic analysis approaches, comparison based testing would appear to hold the most promise for waste disposal models.

It must be realized that it will be virtually impossible to completely verify a program. At best it can be hoped to undertake a series of tests to provide reasonable assurances that the model is correct. It is recommended that a combination of approaches be used for providing AECB with these assurances.

Based on the methods reviewed and their evaluation using the established criteria, the following can be concluded:

1. Desk-checking, walkthroughs and inspections have the advantage of detecting a wide range of possible error types. Desk-checking is less reliable than either inspections or walkthroughs due to the absence of a team;

2. Formal proofs and symbolic verifiers are too theoretical to be of practical use;

3. Cross-reference mappers are reliable and relatively inexpensive. Although specific to a limited number of errors they comprise a high percentage of the total number of errors present;

4. Data flow analyzers are still relatively unproven (moderate reliability) and are fairly expensive. Similar results could be obtained through the use of cross-reference mappers and inspections at less cost;

5. Interface analyzers are designed to detect a very common type of error, are relatively easy to use and are priced moderately;

6. Control structure analyzers do not detect errors per se but can be used to identify program structure;

7. Code auditors can be used to verify adherence to specific standards. The only errors that can be detected are syntax errors that would generally be identified by a compiler;

8. Assertion testing can be used to verify the internal workings of a code. However given that few automated tools are available the level of effort is higher than a walkthrough with only a comparable level of return;

9. Interactive test aids are primarily a tool for error debugging as opposed to error detection;

10. Compiler based tools are easy to use, can reliably detect a variety of common language errors at reasonable cost and low level of effort;

11. Roundoff analyses are too highly specialized and too difficult to use to be generally applicable;

12. Dynamic testing by using input data generators is impractical for waste disposal models because of the difficulty of ascertaining the correctness of the output;

13. Comparison based dynamic testing requires some knowledge regarding the program to ensure that all important functions have been tested. The fact that comparison based testing provides a set of results for verifying the program makes it the most suitable of the dynamic testing approaches for waste disposal models.

Static techniques that are associated with compilers (cross-reference mappers, array index checking, etc.) are commercially available at reasonable cost and possess a fairly high degree of reliability and should be considered. An organized systematic inspection is also an approach that should be considered by AECB. The basis for the inspection should be a checklist of items to be included in the inspection (e.g., array dimensions, subroutine arguments, consistent use of variable types). Dynamic testing, in the form of comparison based testing, is also recommended for adoption by AECB. In spite of the difficulties associated with this approach (e.g., requiring sufficient knowledge to construct test cases to adequately test the program and the existence of a second program/solution for comparison) it is felt that this type of dynamic testing is the most suitable currently available technique.

The test cases should consider such aspects as dominant mechanisms, numerical criteria (time and space discretization), tolerances, iteration limits, convergence parameters, mass balance calculations for water and solute, boundary and initial conditions, non-linear behaviour due to coupled processes, method of calculating auxiliary variables (i.e., velocities), source term behaviour, complete (yet realistic) ranges of input variables.

To summarize, therefore the most effective verification approach for waste disposal models would consist of a combination of different techniques, such as:

1. Full-featured compilers, e.g., with cross-reference mapping, index checking capability;

2. Interface analyzer;

3. Inspection with a checklist for errors not covered by items 1 and 2 (see Appendix A for example errors);

4. Walkthroughs with (preferably) the code author;

5. Comparison based dynamic testing in conjunction with expert opinion used to formulate comprehensive test cases testing all aspects of the program.

The verification requirements placed by AECB on a potential licensee should not be limited to the choice of verification methods as suggested above. Verification should be viewed as an essential Quality Assurance component but not a standalone entity. The verification of a model requires the application of all aspects of a quality assurance program including the development of design and performance specifications for test cases, the assigning of responsibilities to qualified personnel and the archiving of results.

The problems associated with the validation of waste disposal models are in many ways unique to this type of model. Difficulties include the time and space scales of interest, poorly understood processes, inability to collect (and use) sufficient data to ensure site characterization and non-uniqueness of model-observed data fits.

The implications of using a model incorrectly (e.g., when the application and model are incompatible) are severe and are all too often not fully considered in the use of waste disposal models.

State-of-the-art validation approaches all involve matching calculated data with data from either:

1. Laboratory experiments;
2. In situ or field experiments;
3. Natural analogues;
4. Reasoned estimates and bounding analyses.

The use of expert opinion in filling in gaps in existing or forecasted conditions should not be overlooked as a vital part of the total model validation exercise. However, expert opinion should not be viewed as a standalone validation technique.

To assist AECB both in their review of validation exercises and to evaluate model suitability for other applications a checklist of items that should be addressed by any waste disposal model validation has been constructed. These criteria may be thought of in the context of items to be identified as part of the validation exercise. These items have been categorized and are summarized as follows:

1. Comparison data:
   - source of data, e.g., measured, reasoned estimates, bounding analyses;
   - mechanisms/processes represented by data;
   - time dependence, scale, dimensionality;
   - accuracy of data; this highlights the need for well controlled experiments to ensure meaningful results are obtained;
   - uncertainties from unknown/assumed data;
   - empirical coefficients and how they must be evaluated.

2. Model Related:
   - processes represented by model and theoretical basis (conservation of mass, energy, momentum);
   - assumptions of governing equations, e.g., constant density, isothermal, trace concentrations, constant viscosity, equilibrium adsorption;
   - ability of data set to satisfy these assumptions must be demonstrated with the effect of violations quantified;
   - identify restrictions of numerical solution or algorthim, e.g., time and space discretization, tolerances, iteration limits, convergence parameters;
   - restrictions due to hard programmed variables or data libraries.

3. Criteria for Acceptance:
   - are the results logical or at least feasible?
   - visual versus statistically based (i.e., quantitative or qualitative);
   - material balances calculated on a total or individual grid block basis for solute and water;
   - uniqueness addressed?
   - sensitivity of solution to unknown parameters, e.g., to boundary and initial conditions, input variable probability density function;
   - has the effect on the results of data inaccuracies been quantified?
   - do all input variables represent valid (i.e., physically real) parameters?
   - how does the validation comparison compare to the intended application, e.g.,
     i) are the processes the same?
     ii) are the scales the same?

iii) is the quantity of data and level of detail the same for both?

iv) has the impact of differences (e.g., in scale, data requirements) been addressed.

Again the comments made regarding verification as a standalone entity apply to validation. Model validation must be supported by a complete range of quality assurance activities including the creation of design and performance specifications for test cases, the qualifications of the validation team and the documentation function.

Due to the diverse nature of waste disposal models, it is impossible to develop more specific criteria. It is believed that the above criteria provide a general set of guidelines that AECB can use to evaluate validation efforts for virtually all waste disposal models.

The above sections have described the verification procedures and validation checks that are applicable to waste disposal models. Not specifically addressed, however, are other aspects of the Quality Assurance program that should be followed in the verification/validation process. This would include, for example, guidelines and requirements for the development of design and performance test specifications, the documentation procedure (including presentation of results), reviews/audits, problem reports/remedial action and baselining procedures. It is recommended that a review of all quality assurance procedures be undertaken to assist AECB in assessing the reliability of the verification/validation tests conducted by potential licenses.

REFERENCES

Andersson, K., 1985. Prospects of Model Validation Against Field Laboratory Observations. Proceedings NEA Workshop on System Performance Assessments for Radioactive Waste Disposal, Paris.

Andriole, S.J., 1983. Software Validation Verification Testing and Documentation. Petrocelli Books, Princeton, N.J.

ANSI/ASME NQA-1-1986. Quality Assurance Program Requirements for Nuclear Facilities. Sponsored by the American Society of Mechanical Engineers.

Battye, D.J. and M. Parsaei, 1980. The Effect of Computer Dependent Factors when Solving Ordinary Differential Equations. In Production and Assessment of Numerical Software, Academic Press.

Boehm, B.W., 1984. Verifying and Validating Software Requirements and Design Specifications. IEEE Software, January, pp. 75-88.

Budd, T.A., 1981. Mutation Analysis: Ideas, Examples, Problems and Prospects. In Computer Program Testing, edited by B. Chandrasekaran and S. Radicchi.

Fairley, R.E., 1981. Software Testing Tools. In Computer Program Testing. Edited by B. Chandrasekaran and S. Radicchi.

Fischer, K.F., 1974. User's Manual for Code Auditor, Code Optimizer Advisor, Unit Consistency Analyses. TRW Systems Group, Redondo Beach, California, July.

Floyd, R.W., 1967. Assigning Meaning to Programs. Proceedings of the American Mathematical Society Symposium in Applied Mathematics, Vol. 19, pp. 19-31.

Hopkins, T.R., 1980. PBASIC - A verifier for Basic. Software Practice and Experience, Vol. 10, pp. 175-181.

Howden, W.E., 1976. Reliability of the Path Analysis Testing Strategy. IEEE Transactions on Software Engineering, SE-2.

Howden, W.E., 1977. Symbolic Testing and the DISSECT Symbolic Evaluation System. IEEE Transactions on Software Engineering, SE-3.

Howden, W.E., 1978. A Survey of Dynamic Analysis Methods. In Tutorial: Software Testing and Validation Techniques, E. Miller and W.E. Howden, ed., IEEE Computer Society, New York.

REFERENCES

INTERA, 1986. Draft Salt Site Performance Assessment Project Quality Assurance Plan.

INTERCOMP Resource Development and Engineering, Inc., 1976. Development of Model for Calculating Disposal in Deep Saline Aquifers, Parts I and II. USGS/WRI-76-61, PB-256903, National Technical Information Service, Washington, DC.

International Business Machines, 1976. Code Reading: Structural Walkthroughs and Inspections. IPTO Support Group, March.

INTRACOIN Coordinating Group, 1984. International Nuclide Transport Code Intercomparison Study, Final Report Level 1. SKI 84:3, September 1984, Swedish Nuclear Power Inspectrate, Stockholm, Sweden.

Jansen, G.J. and W.M. Hewitt, 1986. International Nuclide Transport Code Intercomparison Study (INTRACOIN) Workshop and Coordinating Committee Meeting. BMI/ONWI-586.

Lantz, R.B.,1971. Quantitative Evaluation of Numerical Diffusion. Society of Petroleum Engineers Journal, September, pp. 315-320.

Miller, E.F., Jr., 1981. Experience with Industrial Software Quality Testing. In Computer Program Testing, edited by B. Chandrasekaran and S. Radicchi.

Miller, W., 1975. Software for Roundoff Analysis. ACM Transactions on Mathematical Software, 1, 2, June.

Moreno, L., I. Neretnieks and C-E. Klockars, 1983. Evaluation of Some Tracer Tests in the Granitic Rock at Finnsjon. KBS TR 83-38.

Myers, G.J., 1979. The Art of Software Testing. John Wiley and Sons, New York.

Osterweil, L.J. and L.D. Fosdich, 1976. DAVE - A Validation Error Detection and Documentation System for Fortran Programs. Software Practice and Experience, 6, pp. 473-486, September.

Panzl, D.J., 1978. Automatic Software Test Drivers. Computer, April, pp. 44-50.

Pickens, J.F. and G.E. Grisak, 1981. Scale-Dependent Dispersion in a Stratified Granular Aquifer. Water Resour. Res., 17(4), pp. 1191-1211.

REFERENCES

Press, W.H., B.P. Flannery, S.A. Teukolsky and W.T. Velterling, 1986. Numerical Recipes - the Art of Scientific Computing. Cambridge University Press, Cambridge.

Ramamoorthy, C.V. and S.B.F. Ho, 1975. Testing Large Software with Automated Software Evaluation Systems. IEEE Transactions on Software Engineering, March, pp. 46-58.

Reeves, M., D.S. Ward, N.D. Johns and R.M. Cranwell, 1985. Theory and Implementation for SWIFT II The Sandia Waste - Isolation Flow and Transport Model for Fractured Media Release 4.84. NUREG/CR-3328, SAND83-1159.

Ryder, B.G. and A.D. Hall, 1975. The PFORT Verifier. Computing Science Technical Report #12, Bell Laboratories, March.

Sandia National Laboratories, 1983. Engineering Procedure Quality Program Requirements for Contractor Furnished Software.

Schwartz, F.W., 1977. Macroscopic Dispersion in Porous Media: The Controlling Factors. Water Resour. Res., 13(14), pp. 743-752.

Sperry Rand Corp., 1979. Sperry Univac Series 1100 Fortran (ASCII) Programmer Reference.

Stucki, L.G., 1973. Automatic Generation of Self-Metric Software. In Proceedings IEEE Symposium Computer Software Reliability, New York, pp. 94-100.

Stucki, L.G. and G.L. Foshee, 1975. New Assertion Concepts for Self-Metric Software. Proccedings, Conference on Reliable Software.

Taylor, R.N., R.L. Merilatt, L.J. Osterweil, 1979. Integrated Testing and Verification System for Research Flight Software - Design Document, NASA CR 159095, July 31.

Teichrow, D. and E.A. Hershey III, 1977. PSL/PSA. A Computer-Aided Technique for Structural Documentation and Analysis of Information Processing System. IEEE Transaction on Software Engineering, SE-3.

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

REFERENCES

Tsang, C-F., 1985. Mass transport in low permeability rocks under the influence of coupled thermomechanical and hydrochemical effects - An overview. Proc. Inter. Assoc. Hydro. Mtg., "Hydrogeology of Rocks of Low Permeability", Part 2, pp. 430-440, Tucson, Arizona, Jan. 7-12.

United States Department of Defense (DOD), 1985. Military Standard Defense System Software Development, DOD STD-2167, June.

Wilkonson, G.F. and G.E. Runkle, 1985. Quality Assurance (QA) Plan for Computer Software Supporting the U.S. Nuclear Regulatory Commission's High Level Waste Management Program. NUREG/CR-4369, SAND85-1774.

Worley, B.A. and F.G. Pin, 1986. Quality Assurance (QA) Plan for the Sensitivity and Uncertainty Analysis Methods Development Project (DRAFT). Prepared for the Salt Repository Project Office by Oak Ridge National Laboratory.

GLOSSARY

ASSERTION TESTING a dynamic analysis technique which inserts assertions about the relationship between program variables into the program code.The truth of the assertions is determined as the program executes.

BENCHMARK a term used to describe a program tested to a certain level or specification. The word is sometimes used to describe the test(s) used in achieving the benchmark (i.e., benchmark tests) or as a verb to describe the process.

BLACK BOX TESTING see FUNCTIONAL TESTING.

BOTTOM UP TESTING a systematic testing philosophy which seeks to test those modules at the bottom of the invocation structure earliest.

BOUNDARY VALUE ANALYSIS a selection technique in which test data are chosen to lie along "boundaries" of input domain (or output range) classes, data structures, procedure parameters, etc. Choices often include maximum, minimum and trivial values or parameters. This technique is often called stress testing.

DYNAMIC ANALYSIS analysis that is performed by executing the program code.

EXHAUSTIVE TESTING executing the program with all possible combinations of values for program variables.

FORMAL PROOF the use of techniques of mathematical logic to infer that a relation between program variables assumed true at program entry implies that another relation between program variables holds at program exit.

FUNCTIONAL TESTING application of test data for the purpose of testing specific program functions without regard to the final program structure.

METRIC a quantitative measure of a software property.

QUALITY ASSURANCE PROGRAM (SOFTWARE) a set of guidelines or procedures that are intended to ensure the correctness of coded software.

STATIC ANALYSIS analysis of a program that is performed without executing the program.

GLOSSARY (cont'd)

STRUCTURAL TESTING a testing method where the test data are derived solely from the program structure.

SYMBOLIC EXECUTION a static analysis technique that derives a symbolic expression for each program path.

TOP DOWN TESTING a systematic testing philosophy which seeks to test those modules at the top of the invocation structure earliest.

VALIDATION the process by which it is established if the model is a correct representation of the physical process it is intended to represent.

VERIFICATION the process by which it is determined if a computer code correctly executes the calculations it is asserted to perform.

WHITE BOX TESTING see STRUCTURAL TESTING.

APPENDIX A

Examples of Common Language Errors

Data declaration

- non-initialized variables, especially array elements
- incorrect type or precision, e.g., due to default typing
- failure to Dimension arrays (COMMON or DIMENSION)
- equivalencing two different variable types (e.g., single to double precision)

Data Reference

- index out of [illegible] (for array)
- index exceeds dimension
- use of undefined (uninitialized) variable
- non-integer index
- computed indexing errors (e.g., use of real number)
- typographical errors
- computed GO TO argument invalid
- invalid do-loop counter or increment

Interface

- inconsistencies between two common blocks or subroutine call and subroutine statement, e.g.:
  i) differing number of varibles
  ii) differing types of variables
  iii) different order of arguments

Comparison

- logical tests between different variable type (e.g., INTEGER vs REAL; single versus double precision)

Input/Output

- use of reserved I/O units
- incorrect file positioning
- inconsistent format/buffer size for input read and file type
- inconsistent volume of data
- no IF (EOF)...statement
- output file line limit exceeded