1/13

# DEVELOPMENT OF AN INTEGRATED SYSTEM COMPUTER CODE FOR REVIEWING THE PERFORMANCE ASSESSMENT OF THE PROPOSED HIGH-LEVEL RADIOACTIVE WASTE REPOSITORY AT YUCCA MOUNTAIN

by
Robert W. Rice[1] and Sitakanta Mohanty[2]
[1]Consultant, 307 Quincy, El Paso, TX 79922, USA, (915) 581-0853
rwrice@aol.com
[2]Center for Nuclear Waste Regulatory Analyses, Southwest Research Institute
6220 Culebra Road, San Antonio, TX 78238, USA, (210) 522-5185
smohanty@swri.org

## ABSTRACT

The Nuclear Regulatory Commission has developed an integrated Total-system Performance Assessment (TPA) code with technical assistance from the Center for Nuclear Waste Regulatory Analyses to evaluate the U.S. Department of Energy Total System Performance Assessment for the proposed high-level radioactive waste repository at Yucca Mountain, Nevada, USA. The TPA code development relied on significant technical contributions and time commitments from a multi-disciplinary team and encompassed a host of considerations ranging from diverse user needs to computer resource requirements. Because the TPA code was developed primarily by scientists and engineers, code development strategies used by computer programmers in the software industry were not fully implemented. Instead, code development strategies evolved during TPA code development. The focus of this paper is on these strategies and the key considerations in planning, developing, and testing this scientific code. The paper presents a brief description of the processes and events modeled in the code and outlines the code structure, which was developed based on user and developer considerations and the implementation of quality assurance procedures considered vital to the acceptability of the code for regulatory review. In presenting the TPA code development process, this paper provides not only insights into the functionality of the TPA code, but also emphasizes good practices that can be used and pitfalls to avoid in developing a large, multidisciplinary scientific code.

## INTRODUCTION

To prepare for reviewing the safety case to be made in the license application by the U.S. Department of Energy (DOE) for a proposed repository at Yucca Mountain (YM), the Nuclear Regulatory Commission (NRC) has developed the capability to conduct independent confirmatory performance assessments (PAs) using the Total-system Performance Assessment (TPA) code (1, 2). As the licensee, the DOE is responsible for conducting a detailed and complete Total System Performance Assessment (TSPA) of the YM repository system adequate to present a safety case in the license application. Rather than duplicating the DOE TSPA, the NRC will use its PA capability, of which the TPA code is a major part, to focus its review of the license application and to analyze or review technical aspects critical to making licensing decisions. The NRC will use the TPA code to check the DOE TSPA results, investigate variants of DOE models, assess repository design changes, and evaluate alternative conceptual models. The TPA code is designed to provide the NRC a versatile tool for exploring a wide range of conceptualizations for the behavior of the YM system by performing calculations that can be viewed either as (i) estimates to check the reasonableness of the DOE TSPA results or (ii) independent

evaluations of aspects of the engineered and natural systems that are highly uncertain, or have the greatest effect on repository performance.

The NRC code development objective is to enable the staff to perform calculations sufficient to implement policy requirements in the public interest in a manner that is professionally competent and financially accountable. The development of a scientific multidisciplinary code, such as the TPA code, can be highly resource intensive and time consuming if issues and steps associated with the development are not clearly defined *a priori*. It is difficult to define all details of a complex and large code development effort *a priori*, however, because of the very large number of models and parameters and, generally, the short time frame available for developing the code. Currently, very little appears to have been published formally in performance assessment literature about the key aspects of a large, integrated scientific code development effort that involves several different professional disciplines undertaking many interacting tasks. Hence, the description of the TPA code development activity[a] presented in this paper may serve as a source of information to developers of other large, integrated scientific computer codes who may learn from these experiences and practices and thereby avoid some of the problems encountered.

This paper provides a brief description of the TPA code components, the code structure, user and developer design considerations, and a review of the code features. Throughout the text, the multidisciplinary aspects of the TPA code development and important features embedded in the code are highlighted.

## TOTAL-SYSTEM PERFORMANCE ASSESSMENT CODE COMPONENTS

In developing the TPA code, the first level of effort was to divide and isolate the total system at a conceptual level into components, or subsystems, so that various parts of the large, complex problem could be addressed by staff with specific areas of expertise. The components of the YM system were defined and grouped into six categories following extensive consensus-building meetings between the NRC and Center for Nuclear Waste Regulatory Analyses (CNWRA) staff. The six components included (i) infiltration, (ii) near-field environment, (iii) waste package (WP) failure and radionuclide releases from the engineered barrier system (EBS), (iv) aqueous radionuclide transport through the unsaturated zone (UZ) and saturated zone (SZ), (v) airborne transport from extrusive volcanic events, and (vi) exposure scenario and reference biosphere. Each component accounted for a process or group of processes.

An overview of these six YM system components is provided in figure 1. As shown in figure 1, precipitation moves throughout the land surface and infiltrates the UZ above the repository, which is divided into subareas for modeling purposes. The infiltration enters the near field of the repository and contacts WPs. WPs, which contain spent nuclear fuel (SNF) and other high-level wastes stored in the repository, may fail by a number of mechanisms including corrosion and the occurrence of seismic, faulting, or igneous disruptive events. After a WP fails, water contacts, dissolves, and transports the SNF in the hydrogeologic units of the UZ and SZ to the discharge point. The discharge point corresponds to the location of the interface between the geosphere and the biosphere where individuals may be exposed to SNF in groundwater. Additionally, individuals may be exposed to radionuclides released from the repository in extrusive igneous disruptive events (not shown in figure 1) that fail WPs and incorporate crushed SNF into magma and eject fine particles that become airborne.

The challenge in modeling these components involved developing mathematical representations of the complex environment at YM. The mathematical model needed to embody complexities associated with (i) long-term behavior of the EBS components, (ii) coupling between the engineered and the natural system components, (iii) large uncertainties in the characterization of geologic components, (iv) flow and transport prediction for highly fractured heterogeneous porous media, and (v) evolution of processes including climate, geology, chemistry, and biosphere over long time scales (~$10^3$–$10^5$ yr).

## TOTAL-SYSTEM PERFORMANCE ASSESSMENT CODE STRUCTURE

The six components of the YM repository site provided the overall framework for developing and implementing modules in the TPA code. Detailed models of these components, which incorporated the aspects of the YM system complexities described in the previous section, were considered to be beyond the scope of the NRC effort. Therefore, models implemented in the TPA code relied on abstractions that allowed calculations to be performed within a reasonable computation time. Model abstractions are simplified mathematical representations of the underlying detailed conceptual model. The TPA model abstractions were verified against the detailed conceptual models and provided an efficient means of analyzing the components of the YM system. The abstractions, including the six components of the YM system and the associated TPA modules, are described in the TPA code user's guide (1).

In addition to modules performing scientific functions, the TPA code consists of five other components: (i) a driver or main program, (ii) an input data reader, (iii) data files, (iv) standalone codes, and (v) a utility module. At the beginning of a TPA simulation, the driver performs initializations, executes the input data reader, which stores all data specified in the primary input file, and controls the sequence of module execution with a subroutine call that has explicit input and output expectations. The input data reader processes data for values that the user is not generally interested in modifying. Standalone codes represent models similar to the subroutines except that they can be executed independently of the driver or the main program of the TPA code. The final component of the TPA code is the utility module. The utility module contains algorithms that are available to all modules, eliminating repeated algorithms. Additional discussions of these components are presented subsequently in this paper.

One of the objectives in the TPA code development was to capture the important aspects of the YM system within the code structure. Additionally, factors such as data analysis requirements and flexibility to accommodate alternative models that will allow the NRC to evaluate the DOE TSPA calculations were critical. The next section describes these factors, which are subdivided into user and developer considerations, that affected both the TPA code development and the module implementation.

## TOTAL-SYSTEM PERFORMANCE ASSESSMENT CODE DEVELOPMENT CONSIDERATIONS AND IMPLEMENTATION

Prior to the TPA code development, the design requirements were established based on experience gained from developing and executing previous versions of the TPA code and after consensus-building meetings between the code developers and the users. The most significant of these developer and user considerations implemented in the TPA code are highlighted in this section. In general, the user considerations relate to the interaction of the users with the TPA code, and the developer considerations are applicable to programming and code development.

## User Considerations

Distinct from developer concerns, user considerations involve execution of the TPA code, accessing subsystem- and system-level results, and providing a perspective about the meaning of the results by supplying an understanding of the flow of information in the TPA code. The user considerations in the TPA code include maintaining transparency and traceability, accounting for the available computer resources of the users, and accommodating multiple users. These considerations are described in the following subsections.

### *Transparency and Traceability*

A code is traceable if there is a complete record through an unbroken chain linking the results unambiguously with the model assumptions and the data used in the computer code (5). A code is transparent if the user can gain a satisfactorily clear picture of the model implementation, the results, and an explanation for the behavior of the results. Transparency and traceability[b] in the TPA code and the interrelationships between the model abstractions are also provided through internal documentation in the source code and information contained in the TPA code user's guide (1). Transparency and traceability attributes are also evident in the example screen output presented in figure 2. The screen message in this figure provides information that the user can review during the TPA simulation to relate final results with key intermediate results. The specific contents of the screen message shown in figure 2 are explained in more detail subsequently in this paper.

### *Computer Resources*

Until recently, detailed modeling of complex systems was limited by the computer resources. Complex systems were often divided into subsystems and simplifying assumptions utilized. However, with recent advances in computer technology, not only is it practical to develop and run codes that place greater demands on computer resources, but user's expectations for computer code capabilities and access have increased. More computer processor speed, memory, and storage are accessible. With these resources, workstations and desktop computers can complete computationally demanding simulations in a reasonable amount of time. The implication for the TPA code development was that the user demanded incorporation of greater process details into the system-level simulation. For example, instead of relying on analytical or semi-analytical solutions that typically use simplifying assumptions, the increased memory and processor speed allow the implementation of numerical models with finer spatial and temporal discretization. Increasing the spatial and temporal discretization, although requiring more memory and processor time, can provide a more accurate numerical solution with fewer simplifying assumptions. The TPA code developers, therefore, had to strike a delicate balance between the user expectations and the available computer resources, including memory, disk space, and simulation-time needs, to ensure the TPA code requirements remained within the limits of these resources for trouble-free code execution.

### *Multiple Users*

One of the challenges in developing the TPA code involved incorporating features needed by multiple code users. During planning for the TPA code development, it was recognized that TPA code users had different objectives and data analysis requirements, and each user had a different degree of familiarity with computer systems. Five general categories of users were identified: (i) analysts familiar with performance assessment who evaluate the system-level performance; (ii) process-level specialists

interested in the behavior of repository subsystems; (iii) managers who perform project management and programmatic functions; (iv) reviewers who are experts in performance assessment at the subsystem or system level; and (v) general users, including the public. Thus, while the code had to fulfill the detail-oriented needs of the specialists, it also had to fulfill the summary-level and easy-to-interpret needs of the nonexperts. Fulfilling such diverse needs resulted in the use of nearly 70 files to present outputs (at subsystem and system level) and required incorporation of numerous user-group-specific features into the TPA code. As an example, process-level specialists were interested in evaluating alternate conceptual models. To provide this capability, code developers implemented provisions for specialists to add such conceptual models through new modules or standalone codes with minimal modifications to the TPA source code. Also, output files were created to provide in-depth information to the process-level specialists. However, for general users, the TPA code utilized summary-level files and screen messages, such as the example shown in figure 2, to provide a transparent and traceable indication of the progress of the TPA simulation through summary-level presentation of important input parameters and intermediate results.

## Developer Considerations

The responsibility of code developers was to correctly implement, test, and document modifications to the TPA code and to complete these tasks on schedule. To assist code developers in accomplishing these tasks, there were several important considerations in the code development procedures, including using a modular design to accommodate new and alternate models, accommodating multiple developers, maintaining consistency in calculations, following quality assurance (QA) procedures, and performing code validation. The following sections describe these considerations.

### *Modular Structure and Implementation*

The TPA code was anticipated to be frequently modified when the repository design changed, new data became available, model abstractions were revised, regulatory compliance criteria were adopted, or output requirements were updated. Thus, an important developer design consideration for the TPA code was to include the capability to readily modify the code. This capability was incorporated into the TPA code by implementing a modular structure. The employment of a modular structure, a standard in software engineering practice (6), permits incremental changes to the code (as new information is available) without disrupting the entire code because most modifications and any subsequent testing are local. Modularity is an important aspect of the TPA code even though it has the disadvantage of producing many more code components. Because of the modular structure, the TPA code can provide the option of replacing a module with an alternative model and comparing results. Aside from developer-related advantages, TPA users and project managers benefitted from the modular structure. A modular structure permits TPA users to focus on individual modules representing subsystems and to evaluate the sensitivity of the results to input data. Additionally, in the development and maintenance activities associated with the TPA code, the modular design assisted the project manager with assigning tasks to developers and testers, monitoring progress, and meeting the code delivery schedule by allowing parallel code development.

Modules were implemented in the TPA code in the form of a subroutine call, standalone code, or table look-up. Because there are various levels of abstraction implemented in the TPA code, selecting the appropriate module implementation or combination of implementations depended on the level of detail in the model abstraction and was related to the source and nature of data, the computational needs in the

module, and requirements for off-line analyses. For example, the standalone code implementation was used for detailed calculations that needed to be conducted both inside and outside the TPA code, and also completed in a reasonable amount of simulation time. If the process-level specialist did not need to perform calculations external to the TPA code, subroutines were implemented to decrease computation time. The table look-up implementation utilized values computed in off-line analyses that required a long simulation time or when tabular data was available.

***Multiple Contributors to Code Development and Testing***

The conceptual models and model abstractions for the six components described in the TPA code components section encompass many technical disciplines including hydrology, material science, seismology, volcanology, geology, climatology, chemistry, geochemistry, health physics, and nuclear engineering. The TPA code developers either directly incorporated models from these different fields into the TPA code or worked together with process-level specialists to write a TPA code module. Although not the most suitable tool for modularization and data access, the FORTRAN programming language was chosen for the TPA code to accommodate the expertise and experience of the majority of these multidisciplinary contributors and to provide continuity with past versions of the TPA code. To the extent possible, all multidisciplinary contributors followed one designated programming style (7) to ensure consistency in source code documentation and input and output data format. For example, although multiple individuals contributed to the TPA code, the implementation was consistent; all TPA output files were consistent in terms of unique identification of file names, date and time of creation, and the code name and version number.

In addition to the multidisciplinary scope of the TPA code, other factors made it necessary to employ special procedures during code development. For example, to efficiently develop the TPA code, meet the schedule requirements, and maintain version control, a project management approach was adopted that employed parallel development using code developers and multidisciplinary contributors. This project management approach allowed many individuals to concurrently develop different modules. In the concurrent code development, module "stubs" were utilized on a temporary basis. Each stub performed the same input and output as the final version of the module and typically performed a simple data manipulation. In this way, code developers were able to concurrently write a module, replace the stub with the new module, and conduct tests.

Many individuals were involved not only in developing source code, but also in testing activities. These activities, combined with the tight development schedule, accentuated the importance of having effective version control. Version control was maintained with Software Configuration and Control System (SCCS) (8) code and other control practices including designation of a code custodian. Although the version control used during TPA code development was not foolproof, it minimized problems associated with updating the source code. Testing occurred throughout the code development cycle with the code developers and multidisciplinary contributors receiving, modifying, and returning the source code to the designated code custodian by following pre-established version control procedures. After a code developer completed the modifications and the source code was tested and ready for incorporation into the controlled version of the code, the developer copied the new file into a designated check in subdirectory and notified the code custodian. Each code developer maintained a check in subdirectory, together with a checkout subdirectory. The checkout subdirectory contained files copied from the controlled version that were being modified. In this way, the designated code custodian was aware of which files were being modified and avoided conflicts arising from one file being simultaneously

changed by more than one individual, which was a potential problem considering the large-scale and the fast-paced development schedule. The designated custodian of the TPA code was responsible for maintaining documentation of the changes in SCCS, testing the changes, documenting differences in results, verifying with the code developer and multidisciplinary contributors that the modifications were implemented correctly, and updating the controlled version of the TPA code.

***Consistency in Calculations***

During TPA code development, considerable emphasis was placed on consistency not only in the programming style and module implementations but also in the algorithms employed to perform the calculations. There are numerous examples in the TPA code of the same type of calculation performed in more than one module. Using similar algorithms in different modules that perform the same calculation is inefficient, and there is a higher probably of introducing errors. For these reasons, it was important to maintain consistency in TPA calculations. To ensure consistency, the TPA code developers centralized algorithms by placing them in a utility module that could be accessed by any of the TPA modules. This required significant coordination with scientists involved in code development. The utility module conducts many generic manipulations including adding, scaling, or copying arrays, transposing matrices, and numerous computations that were specific to the TPA code.

Using utility modules has the added advantage of efficient testing. If an algorithm was contained in many different modules, the algorithm would need to be tested in each module. However, only a single test was required when the algorithm was a part of the utility module. Additionally, for those algorithms employing a numerical solver, the utility module helped to ensure that the same convergence criteria were used for all calculations.

***Quality Assurance Procedures***

The TPA code developers followed the formal QA procedure developed at the CNWRA,[c] which is based on NUREG/BR–0167 (9). Some of the high-level procedural requirements included preparing a software requirement description (SRD), a software development plan (SDP), and a software change report (SCR), and validating the computer code.

The SRD that was prepared after the pre-code-development interactions with process-level experts, documented a general description of the software project, including the technical bases for software development. The technical bases for model abstractions and parameters were developed through numerous interactions between the NRC and CNWRA technical staff. The SRD also provided a vehicle to define software development, changes, and functions, and computational approach. The intended computer platforms and operating systems were also identified in the SRD. Additionally, the mathematical models, control flow, data flow, control logic, user interfaces, and data structure were identified to the extent possible in the SRD.

After completing the SRD, an SDP was prepared that described project plans for conducting the TPA code development effort. The SDP identified new development, modification, maintenance, and the associated activities pertaining to the TPA code software and documentation. The SDP also provided a tool for monitoring software development and presented plans for software configuration management and risk management, including discussions of the standards and testing applied to the software, the necessary resources, and the proposed schedule.

Each change to the TPA code was documented in an SCR that described the specific modifications and provided supporting information. To improve traceability, SCRs were assigned unique control numbers, which were documented within the code. For each change to the TPA code, acceptance testing was performed both at the subsystem and system level. The test cases and test procedures were documented and maintained as a QA record.

The code validation part of the software development activities (i.e., comparison with analytical solutions and/or results generated using numerical tools), which ensures that module abstractions are implemented correctly, was divided into functional testing and "black box" testing. In functional testing, the correct implementation of a specific algorithm or an entire module was evaluated by exercising that portion of the code either as a standalone code, with an program prepared by the tester, or in a TPA code simulation. Black box testing generally involved specifying input values and analyzing the TPA output results for reasonableness and consistency with expected results. As with functional testing, black box testing evaluated results from a single module or from the overall system.

## OTHER CODE FEATURES AND PRACTICES

In previous sections of this paper, good programming practices in the TPA code development, the significant user and developer design considerations, and the methods employed to satisfy those considerations were emphasized. These considerations and features include traceability and transparency, computer resources, needs of multiple users, multiple contributors, module implementation, consistency in calculations, QA procedures, and code validation. In addition to these considerations and features, there were other features that improved users' confidence in the TPA code. These features are discussed in this section and include the treatment of data files, checking the input data for errors, utilizing screen messages, and other useful design and development practices.

To foster transparency, one of the goals of the TPA code developers was to remove hardwired data from the source code and place the data in a primary input file and static data files. The developers carefully distinguished between data that most likely would be used and modified by general users (primary input file) and data that would most likely remain unchanged or modified by process-level experts (static data files). Additionally, some data were identified as not being modified by any users. Because of issues related to schedule, an evolving understanding of the users needs, modifications to the model abstractions, and the large size of the TPA code, some data that might be important to users remains in the source code, and this goal has not been fully achieved. However, in each revision of the TPA code, developers have continued to identify data that might be important to users and move the data out of the source code and into data files.

A useful feature in the TPA code is first- and second-level error checking of the input data. These error checks facilitate debugging and correcting errors by identifying problems during code execution. First-level checks are performed when the TPA code read from data files and ensures the data format is correct; second-level checks are conducted within modules to verify that parameter values were within acceptable ranges. When an error is found, the TPA code execution terminates and a message is written to the screen providing a description of the error.

Screen messages are a transparent way for code developers and users to monitor the progress of a run, perform a preliminary analysis of intermediate results prior to completion of the run, and check whether the run is correctly defined in the primary TPA input file. From the beginning of code

development to the current version of the TPA code, screen message contents have been continuously evolving to provide the most helpful information to code developers and users. Developers and users utilize the screen message as a starting point for high-level debugging to ensure that the code is computing as expected and view some of the important input parameters and results. The screen message from the beginning of a TPA simulation is shown in figure 2 and includes information on important global parameters, paths showing the location of the static data files and standalone codes, disk-space requirements corresponding to selected user options, the largest six radionuclide release rates and time of the release from the EBS, UZ, and SZ in the first subarea and first realization, and the initial results from the second subarea.

Other design features and code development practices are helpful to TPA code developers and users. These features and practices include (i) assigning descriptive names with units in the primary TPA input file to make parameter identification easier (e.g., WellPumpingRateAtReceptorGroup20km-[gal/day] clearly identified this parameter), (ii) utilizing an auxiliary program that automatically generated source code documentation, (iii) providing a user's guide documenting the code capabilities and features (1), and (iv) developing pre- and post-processors that helped the users analyze system- and subsystem-level results.

It is important to acknowledge that the practices presented in this paper evolved during code development. In the TPA code development effort, as in other large projects, there were unforeseen issues and problems. The TPA code developers responded to these issues and problems by implementing new practices. For example, at the initial stages of the TPA code project, code development was performed by a single individual. After four months, however, there were more than five individuals involved in the project. At this time, there were instances when more than one individual modified the same code and considerable time was expended by code developers merging the different versions of the code. In response to this problem, the version control procedures, which were described previously in this paper, were developed and implemented. Another issue identified early in the code development was that the technical contributors needed to use a programming style consistent with the TPA code format. There were instances when the code developers spent considerable time modifying and testing the source code supplied by technical contributors to ensure consistency. In response, code developers emphasized to these contributors the required format for the source code and the importance of providing source code as a module that could be exchanged with the existing code.

## SUMMARY AND CONCLUSIONS

This paper identified several of the good code development practices and features incorporated into the TPA code that will allow the NRC to thoroughly and expeditiously review the adequacy of the DOE safety case for the proposed repository at YM. The practices summarized in this paper evolved during the TPA code development process and became better defined as code development progressed. In this regard, developers of scientific codes will benefit from adopting the strategy presented in this paper.

The flexibility of the TPA code modular design allows the NRC to investigate the DOE model assumptions; alternate conceptual models; and design, site characterization, and compliance demonstration factors. The advantage of the modular approach was that the TPA code modules may be modified and implemented in a timely manner to model the anticipated design changes proposed by the DOE. Additionally, the flexibility of the TPA code development to employ an efficient module implementation was illustrated and includes using subroutines, standalone codes, and table look-ups.

Accommodations were made in the TPA code development process to allow multiple individuals to contribute to the multidisciplinary scope of the TPA code in developing, implementing, and testing modules. Throughout TPA code development activities, remaining within the computer resources of the anticipated TPA code users was a consideration. The needs of users to evaluate subsystem- and system-level TPA output were satisfied by providing documentation and a transparent and traceable flow of results. Utility modules supply the TPA modules with useful algorithms and remove redundant subroutines and functions. With these features, the NRC possesses a tool capable of investigating the DOE safety case for the proposed HLW repository at YM. Moreover, the experiences and practices utilized in developing the TPA code and the description of the TPA code presented in this paper should be useful to TPA code users, as well as the developers of other integrated total-system applications such as the disposal of low- and medium-level waste and other programs.

## ACKNOWLEDGMENTS

The authors wish to thank B. Sagar and W. Patrick for their technical and programmatic reviews of the paper. The process described in this paper involved the participation of many staff members at the NRC and the CNWRA for its implementation; too numerous to recognize individually. Their contributions are gratefully acknowledged. This work was performed by the CNWRA on behalf of the NRC Office of Nuclear Material Safety and Safeguards Division of Waste Management under Contract No. NRC–02–97–009. This paper does not necessarily reflect the views or regulatory position of the NRC.

## REFERENCES

1. S. MOHANTY and T.J. McCARTIN (coordinators), "Total-system Performance Assessment (TPA) Version 3.2 Code: Module Description and User's Guide," Center for Nuclear Waste Regulatory Analyses (1998).

2. S. MOHANTY, T. J. McCARTIN, R. CODELL, R. RICE, R. JANETZKE, M. JARZEMBA, G. WITTMEYER, G. CRAGNOLINO, S. STOTHOFF, P. LAPLANTE, S. HSIUNG, G. RICE, T. AHN, J. WELDY, R. FEDORS, C. McKENNEY, B. HILL, C. CONNOR, J. TRAPP, R. ABU-EID, R. GREEN, B. LESLIE, K. GRUSS, J.A. STAMATAKOS, A. IBRAHIM, N. COLEMAN, J. WINTERLE, A. GHOSH, W. MURPHY, "A Total-system Performance Assessment Code for the Safety Assessment of the Proposed High-Level Nuclear Waste Repository at Yucca Mountain," Eos Transactions/Supplement 80(46): pp. F313–F314 (1999).

3. R.B. CODELL, N. EISENBERG, D. FEHRINGER, W. FORD, T. MARGULIES, T.J. McCARTIN, J. PARK, AND J. RANDALL, "Initial Demonstration of the NRC's Capability to Conduct a Performance Assessment for a High-Level Waste Repository," NUREG–1327, Nuclear Regulatory Commission (1992).

4. R.G. WESCOTT, M.P. LEE, N.A. EISENBERG, T.J. McCARTIN, AND R.G. BACA, eds., "NRC Iterative Performance Assessment Phase 2," NUREG–1464, Nuclear Regulatory Commission (1995).

5. Nuclear Regulatory Commission, "Issue Resolution Status Report—Key Technical Issue: Total System Performance Assessment and Integration," Revision 2 (1999).

6. B. LISKOV and J. GUTTAG, "Abstraction and Specification in Program Development," MIT Press (1986).

7. D.M. COOK, N.H. MARSHALL, E.S. MARWIL, S.D. MATHEWS, and G.A. MORTENSON, "FORTRAN Coding Guidelines," EGG–CATT–8898, EG & G Idaho, Inc. (1990).

8. Sun Microsystems, Inc. "Code Manager User's Guide," Revision A (1992).

9. Nuclear Regulatory Commission, "Software Quality Assurance Program and Guidelines," NUREG/BR–0167 (1993).

10. J.R. WELDY, G.W. WITTMEYER, and D.R. TURNER, "Building Confidence in Quantitative Safety Assessment for Deep Geologic Disposal of HLW: External Peer Review of the NRC Total-system Performance Assessment Code," To be presented at the WM2000 conference (same volume) (2000).

**FOOTNOTES**

[a] This succeeds two previous development efforts reported in Codell, et al. (3) and Wescott, et al. (4).

[b] The level of transparency and traceability in the Total-system Performance Assessment code was verified by an external review group and is documented in Weldy, et al. (10).

[c] Center for Nuclear Waste Regulatory Analyses, "Technical Operating Procedure (TOP)–018" (1999).

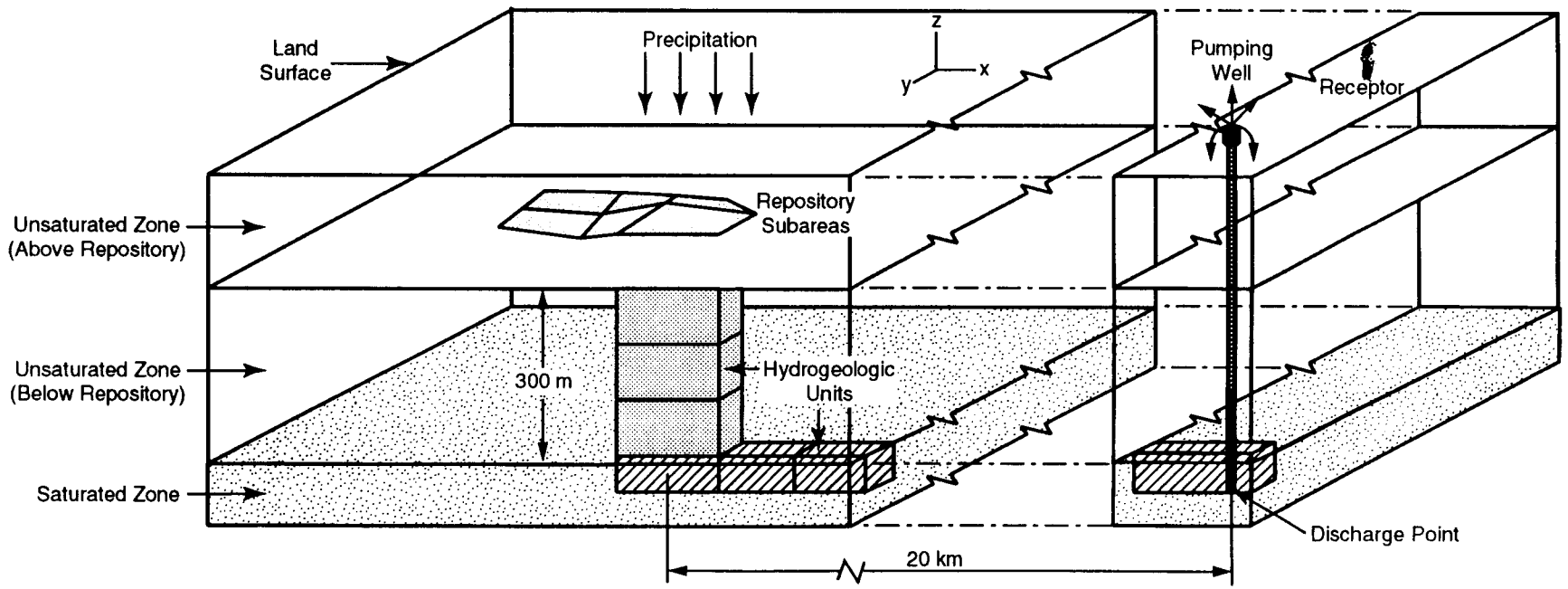


**Figure 1. Overview of the Total-system Performance Assessment conceptualization of the Yucca Mountain repository system showing the unsaturated and saturated zones hydrogeologic units for one of the repository subareas and the pumping well at the receptor location**

```
======================================================================
         exec: Welcome to TPA Version 3.3 PVM capable
       Job started: Fri Jan 21 09:23:04 2000
======================================================================
Specified Global Parameters:

              Compliance Period =  10000.0 (yr)
         Maximum Simulation Time =  10000.0 (yr)
          Number Of Realizations =      1
              Number Of Subareas =      7
              Volcanism scenario =      0 (yes=1, no=0)
               Faulting scenario =      0 (yes=1, no=0)
                Seismic scenario =      1 (yes=1, no=0)
      Distance to Receptor Group =     20.0 (km)


**>>> CAUTION: CHECKING OF NUCLIDES AND CHAINS IS DISABLED <<<**
**>>> You may not be using the standard chains specified   <<<**
**>>> in the invent module.                                <<<**
**>>> (see "CheckNuclidesAndChains(yes=1,no=0)" in tpa.inp)<<<**

The specified path for data  = $TPA_DATA/
The specified path for codes = $TPA_TEST/

**To modify global parameters or the path, stop code execution using control-C**
----------------------------------------------------------------------
    subarea   1 of   7           realization    1 of    1
----------------------------------------------------------------------
exec: calling uzflow
exec: calling nfenv
exec: calling ebsfail
     *** No Corrosion WP Failure ***
exec: calling seismo
exec: failed WPs from INITIAL   event =     16 at time =      0.0 yr
 *** failed WPs:           16  out of         1663 ***
exec: calling ebsrel
                Highest release rates from Sub Area           1
             Tc99    1.6936E-02 [Ci/yr/SA] at  2.198E+03 yr
             Ni59    3.5837E-03 [Ci/yr/SA] at  3.076E+03 yr
             C14     1.8156E-03 [Ci/yr/SA] at  3.076E+03 yr
             Cs135   6.5995E-04 [Ci/yr/SA] at  4.191E+03 yr
             Se79    6.4718E-04 [Ci/yr/SA] at  2.198E+03 yr
             I129    5.1619E-05 [Ci/yr/SA] at  2.251E+03 yr
exec: calling uzft
                    Highest release rates from UZ
             Tc99    1.6791E-02 [Ci/yr/SA] at  2.251E+03 yr
             Ni59    3.5555E-03 [Ci/yr/SA] at  3.384E+03 yr
             Cs135   6.5933E-04 [Ci/yr/SA] at  4.191E+03 yr
             Se79    6.4038E-04 [Ci/yr/SA] at  2.251E+03 yr
             I129    5.1098E-05 [Ci/yr/SA] at  2.307E+03 yr
             Cl36    2.0831E-05 [Ci/yr/SA] at  2.251E+03 yr
exec: calling szft
                    Highest release rates from SZ
             I129    4.3106E-05 [Ci/yr/SA] at  8.897E+03 yr
             Cl36    1.7282E-05 [Ci/yr/SA] at  8.101E+03 yr
     The remaining          18  nuclide(s) have zero release
----------------------------------------------------------------------
    subarea   2 of   7           realization    1 of    1
----------------------------------------------------------------------
exec: calling uzflow
exec: calling nfenv
exec: calling ebsfail
```

**Figure 2. Example of a screen message from the Total-system Performance Assessment code.**