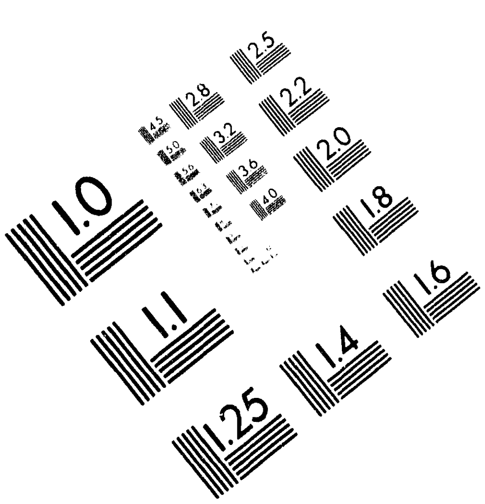
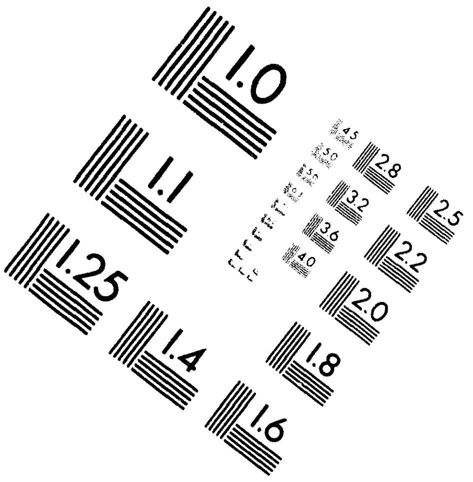
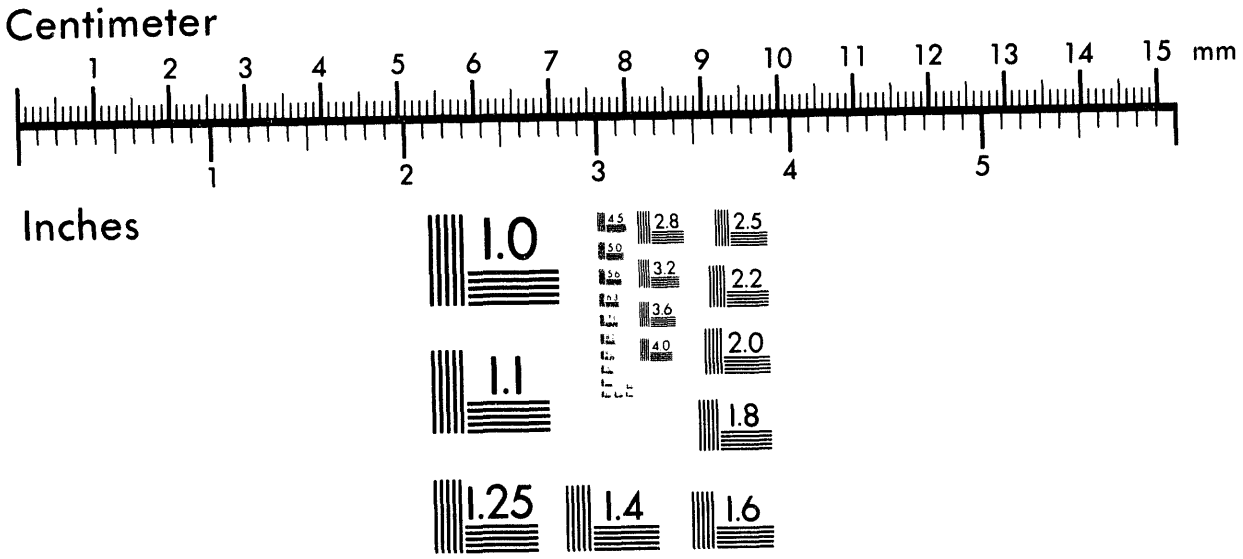
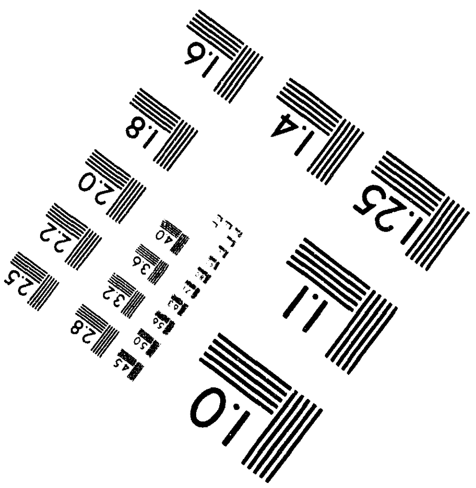
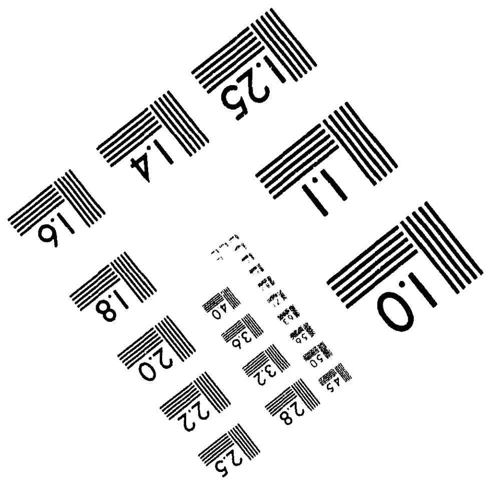



AIIM

Association for Information and Image Management

1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910
301/587-8202



MANUFACTURED TO AIIM STANDARDS
BY APPLIED IMAGE, INC.

1 of 1

ON INTEGRATING MODELING SOFTWARE FOR APPLICATION TO TOTAL-SYSTEM PERFORMANCE ASSESSMENT

Lynn C. Lewis
Lawrence Livermore National Laboratory
Mail Stop L-206
P.O. Box 808
Livermore, CA 94550
(510)422-8949

Michael L. Wilson
Sandia National Laboratories
Mail Stop 1326
P.O. Box 8500
Albuquerque, NM 87185
(505)848-0770

ABSTRACT

We examine the processes and methods used to facilitate collaboration in software development between two organizations at separate locations—Lawrence Livermore National Laboratory (LLNL) in California and Sandia National Laboratories (SNL) in New Mexico. Our software development process integrated the efforts of these two laboratories. Software developed at LLNL to model corrosion and failure of waste packages and subsequent releases of radionuclides was incorporated as a source term into SNL's computer models for fluid flow and radionuclide transport through the geosphere.

I. INTRODUCTION

The Yucca Mountain Site Characterization Project (YMP) of the U.S. Department of Energy is studying Yucca Mountain, Nevada, as a potential site for permanent disposal of high-level radioactive waste. The second iteration of total-system performance assessment for the site was conducted in 1993 and is referred to as TSPA-93.[1] In TSPA-93, computer models were used to evaluate the performance of a potential Yucca Mountain repository in terms of cumulative release of radioactivity to the accessible environment and in terms of individual drinking-water dose.

This paper describes some of the software development undertaken for TSPA-93, particularly methods to maximize use of existing computer models and preserve established working relationships between modelers and software developers. These methods were used to assemble application-specific software systems from a set of existing and evolving components or software modules. Modules continually evolved to balance model sophistication against operational constraints (e.g., execution time and memory limitations) and in response to new site characterization information. Rapid changes to the set of modules required methods for concurrently assembling, testing, and installing multiple application-specific software system releases. Integrating modules from different developers written in different programming languages placed increased demands on Software Configuration Management (SCM) methods. We address the specific SCM measures we took to identify and track the relationship between modules and the software releases they belong to.

II. MULTIPLE PROGRAMMING LANGUAGES

In developing the TSPA-93 software components, we exploited the advantages of computer languages best suited to each component's or module's needs. In this way we were able to pursue new development using state-of-the-art programming practices, such as object-oriented programming, while preserving the value of investments in established, tested computer programs written in older languages. It was possible to quickly and reliably construct a large TSPA-93 computer model that incorporated a newly developed LLNL model of radionuclide release from the engineered barrier system (EBS) of a potential repository, written in an object-oriented programming language (C++), with SNL's groundwater-flow and radionuclide-transport models, written in Fortran 77. There was no need to perform costly programming language conversion to construct the TSPA-93 code. The SNL models are the product of a well established software effort

[illegible]

extending over several years, and they are comprised of tens of thousands of lines of Fortran.

The LLNL EBS model is based on the LLNL-developed Yucca Mountain Integrating Model (YMIM) code.[2,3] The SNL code includes the programs TOSPAC (Total System Performance Assessment Code) and WEEPTSA (Fracture Weep Total System Analyzer). In cooperation with SNL, LLNL produced an interface that allowed YMIM to be called by TOSPAC and WEEPTSA. We implemented the Fortran to C++ calling protocols by declaring the top-level YMIM modules as C language function calls using the C++ extension to the "extern" declaration.[4] This tells the compiler to use the C language naming conventions, allowing normal C–Fortran interface operation.

A. TOSPAC

TOSPAC models isothermal groundwater flow through the unsaturated zone and radionuclide transport through layered, fractured, porous media. [5,6] Saturated-zone transport can also be modeled. TOSPAC models flow and transport in only one spatial dimension because it is intended to be used in a stochastic mode, in which many realizations of the system are simulated using the Monte Carlo method.[7] Probability distributions are defined for key input parameters, and a distribution of output releases is generated using multiple TOSPAC runs. Each Monte Carlo realization represents a possible future history of the potential repository system.

To take into account stratigraphic variability across the potential repository, the repository area is divided into eight subregions, each of which is modeled as a one-dimensional, vertical column. To further take into account variability of waste-package environments within the subregions, the waste packages for each TOSPAC run are divided into ten groups, with temperature and water-contact mode varying from group to group. YMIM is called from TOSPAC to calculate container failure and radionuclide releases, with YMIM keeping track of the ten container groups in parallel.

B. WEEPTSA

WEEPTSA models an alternative conceptual model of unsaturated-zone groundwater flow and transport in which flow is assumed to take place entirely by locally saturated flow within a probabilistically defined fracture network.[1,8] Like TOSPAC, WEEPTSA is used in a stochastic mode with the Monte Carlo method. The calculation of flow and transport is simplified, allowing the calculation of releases from waste packages to be more complex. Each waste package is tracked individually by WEEPTSA and has its own history of temperature and water contact. Rather than tracking all waste packages in parallel, as is done by TOSPAC/YMIM, container failure and radionuclide releases are calculated sequentially, with WEEPTSA calling YMIM for each waste package in turn.

YMIM is flexible enough to be used in two very different ways by TOSPAC and WEEPTSA, thanks in part to the modular structure provided by C++.

C. YMIM

YMIM is an integrated model of the EBS. It was developed to help identify research areas that should be emphasized.[3] YMIM is highly modular so that a model of an individual process can be easily modified or replaced without interfering with the models of other processes.

The EBS includes the containers for the waste packages, the waste form, and the emplacement mode of the waste packages. YMIM evaluates the waste-isolation capacity of the EBS by modeling these components and estimating the EBS response to physical and chemical parameters.

Both model development and parameter estimation can be costly and time-consuming. To address these problems, YMIM is used to identify areas where YMP research resources should be placed. Specifically, it helps identify components and processes for which simple models or approximate estimations are adequate, and those for which more sophisticated models and accurate estimates are required.

YMIM models radionuclide releases from a set of waste packages containing spent nuclear fuel, all of which are subject to similar temperature, hydrologic, and geochemical conditions. Although all of the packages are subject to similar conditions, there is variability in their releases because it is not assumed that they fail at the same time. The packages fail at different times and release nuclides at different rates because of statistical variability in maximum corrosion rates between packages. Several YMIM runs are required to model all of the packages in a potential repository, which would be subject to a range of temperatures, hydrologic conditions, and geochemical conditions.

III. CONFIGURATION MANAGEMENT

Software Configuration Management (SCM) is central to successful software development by contributors operating at different locations. The LLNL source-term software developers use the UNIX Source Code Control System (SCCS) as the low-level source code database manager to maintain all text files associated with the project.[9] In addition to source code, these files include test data, test results, and release-configuration tag files. Release-tag files define the set of modules and revision levels comprising a software version or code release.

SCM is a software development management process. It helps ensure traceability of software functionality to the requirements established by the user/modeler. SCM provides a way of linking software development activities and their resultant products. Such linking means that each code release can be defined in terms of the set of its constituent elements or configuration items at specific revision levels. The configuration items include documents, test cases, and source-code modules.

The primary issue addressed by SCM for TSPA-93 was concurrent development of both near-term and future code releases. Concurrent development requires the traceability of the module-revision-level composition of each TSPA-93 release. For example, LLNL software developers working on the EBS modeling code had to implement "quick-fix" changes to a radionuclide release source-term module already deployed in the TSPA-93 code at SNL while continuing longer-term YMIM development. Quick fixes were typically performed to resolve issues arising from differences in the operating system and compiler versions being used at LLNL and SNL, and from computer memory and time limitations.

We used UNIX SCCS and electronic mail to implement SCM for TSPA-93 code development. The UNIX SCCS is a tool for managing changes to text files.[9] These files are typically source code but may include test data and documentation, i.e., any element of a software system that can be represented as a text file.

SCCS acts as a custodian of text files and can:
(1) Store, update, and retrieve any version of a file.
(2) Control privileges for access to files.
(3) Identify the version of a retrieved file.
(4) Record who modified a file, and when, where, and why changes were made.

These files are the configuration items that make up a code release. Text strings serving as identification keywords or ID keys are incorporated into a file by SCCS to label retrieved files with identifying information, including module name, time stamp, revision level, and application/use location. ID keys form the basis for tracking revision levels to establish traceability of software functionality for specific code releases.

EBS model source code was delivered to SNL via E-mail on Internet as ASCII-encoded, archive-formatted files. The UNIX Tape Archive Utility (TAR) was used to generate the archive-formatted files.

Our SCM approach used two primary methods. First, concurrent multiple lines of descent for module revisions were tracked. Second, each module source code was tagged to identify its release configuration.

A. Multiple Lines of Descent for Modules

Supporting multiple concurrent TSPA-93 code releases required that module source code be allowed to evolve along multiple and concurrent lines of development, descending from common ancestor versions. These branches are necessary when a previously deployed TSPA-93 release needs a modification specific to that release and not applicable to or practical to incorporate in any other development line. SCCS branching facilities were exploited to support this requirement. Figure 1 shows the relationship between module revisions and the code releases they affect. Note that code release 1.1 uses revision 1.1.2 of module$_1$ avoiding unneeded features of revision 2.0.

B. Release Configuration Tagging

Release configuration tagging uses SCCS ID keys to link each module to the specific code release configuration it participates in. This link makes it possible to reconstruct an "as-built" release

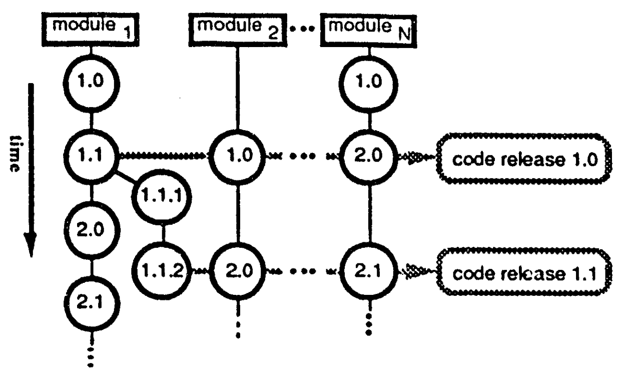


Figure 1. The relationship between module versions and code releases.

configuration to allow continuation of release/application specific development. A release-tag database defining the set of modules and revision levels comprising a code release was implemented. This database extends SCCS capabilities, providing full software-release configuration-management capabilities.

IV. CONCLUSION

The collaboration between LLNL and SNL was successful, and resulted in a better performance-assessment modeling code than either group could have produced independently. It is crucial that national projects like the YMP distribute software-development activities to the research and engineering centers possessing the necessary modeling expertise. Distribution allows software developers and end-users to collaborate with greater efficiency than would be possible if software were developed centrally or at one of the modeler's locations. Furthermore, needless duplication of similar or identical software-development tasks can be virtually eliminated. The ability to transfer computer files back and forth over the Internet is an essential element in maintaining efficient long-distance collaborations.

ACKNOWLEDGMENTS

The authors acknowledge the contribution of the other programmers involved in the TSPA-93 LLNL/SNL integration effort: Jim Gansemer of LLNL and Jack Gauthier of SPECTRA Research Institute. The work could not have been accomplished without them. We wish to acknowledge the helpful suggestions of Bill O'Connell and his review. We also appreciate the editorial assistance of Jay Cherniak. This work was supported by the Performance Assessment Task (WBS 1.2.5.4.1 and 1.2.5.4.2) of the Yucca Mountain Site Characterization Project. Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

REFERENCES

1. M. L. Wilson, J. H. Gauthier, R. W. Barnard, G. E. Barr, H. A. Dockery, E. Dunn, R. R. Eaton, D. C. Guerin, N. Lu, M. J. Martinez, R. Nilson, C. A. Rautman, T. H. Robey, B. Ross, E. E. Ryder, A. R. Schenker, S. A. Shannon, L. H. Skinner, W. G. Halsey, J. Gansemer, L. C. Lewis, A. D. Lamont, I. R. Triay, A. Meijer, and D. E. Morris, *Total-System Performance Assessment for Yucca Mountain—SNL Second Iteration (TSPA-1993),* Sandia National Laboratories, Albuquerque, NM, SAND93-2675 (1994).

2. J. Gansemer and A. D. Lamont, *User's Guide to the Yucca Mountain Integrating Model (YMIM),* Lawrence Livermore National Laboratory, Livermore, CA, UCRL-LR-116446 (1994).

3. A. D. Lamont and J. Gansemer, "Integrated Modeling of Near Field and Engineered Barrier System Processes," in *Proc. Fifth Annual International Conference on High Level Radioactive Waste Management,* American Nuclear Society, La Grange Park, IL, 1994.

4. B. Stroustrup, *The C++ Programming Language,* 2nd ed., pp. 524–525, Addison-Wesley, Reading, MA (1991).

5. A. L. Dudley, R. R. Peters, J. H. Gauthier, M. L. Wilson, M. S. Tierney, and E. A. Klavetter, *Total System Performance Assessment Code (TOSPAC), Volume 1: Physical and Mathematical Bases,* Sandia National Laboratories, Albuquerque, NM, SAND85-0002 (1988).

6. J. H. Gauthier, M. L. Wilson, R. R. Peters, A. L. Dudley, and L. H. Skinner, *Total System Performance Assessment Code (TOSPAC), Volume 2: User's Guide,* Sandia National Laboratories, Albuquerque, NM, SAND85-0004 (1992).

7. M. L. Wilson, F. C. Lauffer, J. C. Cummings, and N. B. Zieman, "Analyzer for Performance Assessment of Yucca Mountain," in *Proc. Second*

Annual International Conference on High Level Radioactive Waste Management, American Nuclear Society, La Grange Park, IL, 1991.

8. J. H. Gauthier, "An Updated Fracture-Flow Model for Total-System Performance Assessment of Yucca Mountain," in *Proc. Fifth Annual International Conference on High Level Radioactive Waste Management*, American Nuclear Society, La Grange Park, IL, 1994.

9. *Programming Utilities for the Sun Workstation*, Sun Microsystems, Inc., Mountain View, CA (1985).