BUILDING QUALITY INTO PERFORMANCE AND SAFETY ASSESSMENT SOFTWARE

L.C. Wojciechowski
Atomic Energy of Canada Limited, Whiteshell Laboratories
Pinawa, Manitoba, Canada

ABSTRACT

Quality assurance is integrated throughout the development lifecycle for performance and safety assessment software. The software used in the performance and safety assessment of a Canadian deep geological repository (DGR) follows the CSA quality assurance standard CSA-N286.7 [1], Quality Assurance of Analytical, Scientific and Design Computer Programs for Nuclear Power Plants. Quality assurance activities in this standard include tasks such as verification and inspection; however, much more is involved in producing a quality software computer program. The types of errors found with different verification methods are described. The integrated quality process ensures that defects are found and corrected as early as possible.

1. INTRODUCTION

The deep geological repository (DGR) project software was originally established to assess the environmental and human health impacts of the concept for disposal of Canada's nuclear fuel waste. In the past, the safety of a Canadian DGR for used fuel was presented in the Atomic Energy of Canada Limited (AECL) Environmental Impact Statement [2,3], AECL Second Case Study [4], Ontario Power Generation (OPG) Third Case Study (TCS) [5], OPG TCS/Horizontal Borehole Case Study [6] and the Nuclear Waste Management Organization (NWMO) Glaciation Scenario safety assessement [7]. Currently other case studies are in preparation by the NWMO in support of the Adaptive Phased Management (APM) approach for dealing with Canada's nuclear fuel waste.

This paper focuses on the project software consisting of an executive program, SYVAC3 (Systems Variability Analysis Code generation 3), a computer model representing the hypothetical reference system specified for a case study of the disposal of nuclear waste (CC4 – Canadian Concept generation 4) and a set of routines required by the CC4 model, which may be applicable to other modeling problems, called ML3 (Modelling algorithm Library – generation 3).

The SYVAC3-CC4 code is being maintained in compliance with the relevant quality assurance standard CSA N286.7-99 [1]. Building quality in the software involves many integrated activities. The framework for Software Quality Assurance (SQA) activities, as documented in Schulmeyer and McManus [8], includes the following steps: requirements, design, implementation, verification, and operation. Each of these steps has one or more verification activities. Examples of verification activities include

- visual inspection,
- software walkthroughs,
- static tests completed by software tools, and

- dynamic tests such as integration testing and system testing.

2. VERIFICATION ACTIVITIES

Several methods are employed during the software development lifecycle to build quality into the system. Each type of testing (e.g. integration and system) serves a purpose and finds different defects [9]. Verification activities, in particular, reduce the amount of defects in the final product by detecting and preventing the defects before they get to the release version.

The verification activities for the following steps will be discussed in the subsequent sections: requirements, design, implementation, integration testing, verification, and operation. Validation activities also play an important role. Several methods are used to overcome the difficulties in validating over long time scales.

3. REQUIREMENTS

During the requirements phase, the analyst documents the functional requirements for the programmer to use in creating the design and for testers to use in testing the code. The requirements typically include a description of the function, a description of data flow, equations and logic defining the functions, definitions for the mathematical symbols used, and possibly test data. Defects may be detected in any of these documents by independent visual inspection and/or software walkthroughs.

Defects typically identified include

- incorrect data flows,
- physical unit imbalances in equations,
- incorrect equations,
- incorrect boundary or initial conditions, and
- incorrect or undefined mathematical symbols.

4. IMPLEMENTATION

During the implementation step, several verification techniques are used. Implementation includes the production of both the design documents and source code. The design documents are generally produced from the requirements specifications and include module design specifications, data dictionaries, file format descriptions and structure charts. When the design documents are produced from the requirements specifications, this provides another check on the requirements specifications. If defects are found they can be passed back to the analyst for correction.

Currently in the design phase, software tools are used to generate and check some of the design documents, which aid in decreasing the amount of defects in the resulting product. The design documents are written in HTML and are linked together for easy access to the programmer or user of the code. A broken link checker, a freeware tool, is used to check that the links are correct. A third party software tool called Understand for FORTRAN is used to automatically generate structure charts.

Source code is generated from the design documents, verifying the design. In the coding phase both manual and automated verification techniques are used. An independent visual inspection and/or software walkthrough are examples of manual techniques. Several static analysis tools are used for automated verification techniques. These tools check if the source code follows standards defined for the project and if physical units of variables in equations and expressions are properly balanced. The static analysis tools, which were created in-house, are

- DDGEN/INSDEF – DDGEN creates a data dictionary from the source code. INSDEF automatically inserts variable definitions from the data dictionary in to the variable declaration sections of the FORTRAN code. These tools ensure consistency of definitions throughout the code and corresponding data dictionary.
- CHEKER – FORTRAN format checker parses the code and coding standards violations such as lines extending beyond column 72, variable declarations not in alphabetical order, and incorrect indentation of blocks of code.
- UNITCK -
- - Units Auditor uses FORTRAN source code and data dictionaries to detect unit imbalance. The types of defects checked are missing units, units not in the main database, unit definition mismatches between the code and data dictionary, unit imbalances in expressions, and coding standards violations. Unit checks in expressions include comparison of the left side of an assignment to the right side of an assignment, and comparison of logical expressions.

The visual inspection and tools mentioned above provide a static analysis of the source code. A dynamic analysis is completed by executing the source code.

In the coding phase, the code that has been revised is executed using the data from the requirements step and the results are compared to previous results to ensure any changes produce the expected effect. The debugger may be used at the implementation step to check intermediate results. The debugger aids in the quality process by helping the tester find issues without modifying the code with items such as print statements that are not removed properly and end up in the final product. Integration testing is completed as part of the implementation step.

Defects typically found during the implementation step include

- incorrect product version numbers,
- inconsistencies between requirements specifications, design and code such as equations present in the requirements that are not implemented in the design and code,
- missing or incorrect equations,
- failure to meet coding standards,
- incorrect physical units in equations or variable definitions, and
- spelling mistakes or incorrect references to requirements.

5. INTEGRATION TESTING

During the integration step, the newly implemented code is combined with the existing code to create a new complete program. The integration tester sets up and executes the run with corresponding input files and the implementor will check if their code changes run as expected.

In addition to tracking source code and documentation, database files are also maintained in a strict manner. When new data is added or current data is modified, an independent visual inspection of the file is completed and the reviewer and date are documented within. The database is treated similar to the source code at integration. The new files are combined with the existing database to create a new input file, which can be used with the corresponding code change during the integration test.

An integration test document is produced to keep track of the tests run. It lists the code tested, the input files used, tests run, and any comments on the test results.

Defects found at this step include

- unexpected results,
- spelling mistakes or other errors in documentation,
- incomplete source code changes,
- data out of valid ranges, and
- incorrect array dimensions.

6. SYSTEM TESTING

Once all the source code changes are made, they are tested as a whole with the existing program. In integration testing, each individual change was verified. In system testing, all changes are checked together as one code change may affect another. The system testing activities include creating a test plan, executing the test plan, and writing documentation. Test cases are generated to test the code changes implemented. Test cases can also be written to see if other changes outside the system, such as moving to a different computer or compiler, have an effect on the code. Information for setting up the test cases is gathered from the software development plan, change requests, requirements specifications, and implementation documentation. The test/change relationship is defined in a matrix relating each change to a test case. The matrix is developed so anyone examining the tests can determine, at a glance, which changes are exercised in a particular test case.

Each test case has a test case identifier, tester name, purpose/outline, methodology, input data, criterion for success, and results/analysis. The expected results do not have to be written in great detail as the volume of output is huge for any of the DGR studies. However, expected results must be understandable by others reviewing the test cases and measurable so a pass/fail status can be determined. This information is provided in the test plan, which is reviewed before testing takes place. Test results are added as the tests are executed.

The test code is copied to test directories, compiled, and linked to maintain independence from the previous development steps and to keep a record of what configuration was actually tested.

Each test case is executed in turn by a tester who did not participate in writing the source code. The tester compares the actual results to the expected results. If an error is found, it is sent back to the previous step for correction. Once the error is corrected, another version of the test executable is made and the test is re-run. Depending on the severity of the error, all tests may need to be re-run with the new executable.

Once all the tests are run, the results are added to the test plan creating the verification test report. Each test is documented with the defects found. A reviewer checks each test case. A summary of defects found and their disposition is written in the verification testing report. Defects found at this step are similar to those found during the integration testing. Once the defects found have been corrected and the code re-tested, the code is pronounced ready for release to the installation step.

7. INSTALLATION

During the installation step the source code and documentation are copied to the reference directories. Command scripts are used to document where the files came from that are installed. Also, the source code is compiled and a reference case is run and compared to the last integration test to ensure that all the code has been copied correctly. Once the products are ready for installation into reference directories, there still may be defects such as

- missing requirements specifications, design products, or source code, and
- incorrect version numbers.

The products with defects are passed back to the appropriate step for correction. In general, the verification techniques in the previous steps ensure that minimal defects reach this step.

8. OPERATION

Once the code is released and ready for use, the verification does not end as more visual inspections take place during operation or application of the software. When the users are running the code, several verification steps are taken as well. First, a case checklist is created to ensure all settings for the code and proper input files are used. This checklist is reviewed to ensure it is what the user requires. To run the code, the user creates a batch file, which serves as a record of what files were used in the run and allows for repeatability without introducing error. A log file is created during the execution of the code to document what was actually run. As the code is executing, the user monitors it to ensure there are no unexpected events. The user checks the log files and output files to determine if the case completed without crashing. Any warning or error messages are examined to see if they are expected or need further investigation.

9. VALIDATION

In the previous sections, attention was given to verification activities that are performed at the various software development steps to ensure the software products of a given step fulfill the requirements of a previous step. Validation is performed to confirm a model provides an acceptably accurate, detailed, and understandable representation of the phenomena of interest, for the intended interpretation and application of the results obtained with it. Validation activities are completed periodically. The models for long-term behavior of deep geologic repositories, such as those used in the SYVAC3-CC4 code, cannot be fully validated as defined in CSA N286.7-99 [1]. Several methods are used to overcome the difficulties in validating over

long time scales. A variety of tests can be performed that provide partial validation. In the past, code intercomparsion studies were completed with other similar codes. The status of the verification and validation history of SYVAC3-CC4 is summarized in Garisto and Gierszewski [10]. Garisto et al. [11] describes a related system model test that is based on a comparison with results of the Swedish SR97 safety assessment [12]. Goodwin et al. [13] describes four additional validation tests. Also, in the course of recent case studies, the SYVAC3-CC4 model has been compared against results from FRAC3DVS [5, 6, 7]. The entire system model cannot be validated as a whole but the individual models, as shown in the examples above, have been validated where possible.

10. IDENTIFICATION AND TRACEABILITY

In large software projects, spanning many years, such as that used for the DGR project, good practices employed in the identification of each software component and good practices in configuration management make configurable items more traceable and usable. It is not uncommon to go back through previous versions of models to check on how a particular function or requirement was implemented, or if it was implemented at all. The user may want to check how the code behaved during a previous test. If every configurable item is well-labeled and stored properly, the users can find what they are looking for easily, and should be able to repeat a verification test within reason as computer systems change over time. A good naming convention for files is a must, as well as strict adherence to the naming convention by all project members.

In the age of the graphical user interface (GUI), point and click operations are not necessarily always recordable. The creation of command files to log operations to maintain a record of actions is good practice. Also the creation of a log file during operation is beneficial. These command files or log files can serve as a record or can be used to repeat operations especially when the codes are used for multiple projects over many years.

11. CONCLUSIONS

A wide variety of verification techniques or activities have been used through all stages of software development for the software used in the performance and safety assessment of a Canadian deep geological repository. Verification activities have reduced the defects in the final product by detecting and preventing the defects from reaching the release version.

Since it is not possible to compare predictions with observations for performance and assessment software, good quality assurance methods, in particular verification, play an important role in the detection and prevention of defects. Also, good practices used in identification of each software component, configuration management, and operation of the code make for good traceability and usability.

ACKNOWLEDGEMENTS

The author would like to thank the NWMO for funding this work. The author would like to acknowledge the contributions of past and present researchers who have assisted with the development of the performance and safety assessment software mentioned within this paper.

REFERENCES

[1] CSA (Canadian Standards Association). 1999. N286.7-99, Quality Assurance of Analytical, Scientific and Design Computer Programs for Nuclear Power Plants. Canadian Standards Association Standard.

[2] Atomic Energy of Canada Limited (AECL). 1994. Environmental impact statement on the concept for disposal of Canada's nuclear fuel waste. Atomic Energy of Canada Limited Report, AECL-10711, COG-93-1. Chalk River, Canada.

[3] Goodwin, B.W., McConnell, D., Andres, T., Hajas, W., LeNeveu, D., Melnyk, T., Sherman, G., Stephens, M., Szekely, J., Bera, P., Cosgrove, C., Dougan, K., Keeling, S., Kitson, C., Kummen, B., Oliver, S., Witzke, K., Wojciechowski, L., Wikjord, A. 1994. The disposal of Canada's nuclear fuel waste: Postclosure assessment of a reference system. Atomic Energy of Canada Limited Report AECL-10717, COG-93-7. Pinawa, Canada.

[4] Goodwin, B.W., Andres, T., Hajas, W., LeNeveu, D., Melnyk, T., Szekely, J., Wikjord, A., Donahue, D., Keeling, S., Kitson, C., Oliver, S., Witzke, K., Wojciechowski, L. 1996. The Disposal of Canada's Nuclear Fuel Waste: A study of postclosure safety of in-room emplacement of used CANDU fuel in copper containers in permeable plutonic rock. Volume 5: Radiological Assessment. Atomic Energy of Canada Limited Report AECL-11494-5, COG-95-552-5. Pinawa, Canada.

[5] Garisto, F., Avis, J., Calder, N., D'Andrea, A., Gierszewski, P., Kitson, C., Melnyk, T., Wei, K., Wojciechowski, L. 2004. Third Case Study - Defective Container Scenario. Ontario Power Generation, Nuclear Waste Management Division Report 06819-REP-01200-10126-R00. Toronto, Canada.

[6] Garisto, F., Avis, J., Calder, N., Gierszewski, P., Kitson, C., Melnyk, T., Wei K., Wojciechowski, L. 2005. Horizontal borehole concept case study. Ontario Power Generation Report 06819-REP-01200-10139-R00. Toronto, Canada.

[7] Garisto, F., Avis J., Chshyolkova, T., Gierszewski, P., Gobien, M., Kitson, C., Melnyk, T., Miller, J., Walsh, R., Wojciechowski, L. 2010. Glaciation Scenario: Safety Assessment for a Deep Geological Repository for Used Fuel. Nuclear Waste Management Organization Report NWMO-TR-2010-10. Toronto, Canada.

[8] Schulmeyer, G.G., McManus, J.I. Handbook of Software Quality Assurance. Third Edition. 1999. Prentice Hall PTR. Upper Saddle River, NJ, 07458 USA.

[9] Wojciechowski, L.C., Ramsay, J.R., Sherman, G.R. 1995. Verification of Scientific Modelling Software. Proceedings of the Summer Computer Simulation Conference 1995.

[10] Garisto, F., Gierszewski P., 2001. Summary of Verification and Validation Studies for SYVAC3-CC4 and its Submodels. Ontario Power Generation Nuclear Waste Management Division Report 06819-REP-01200-10043-R00. Toronto, Canada.

[11] Garisto, F., Gierszewski, P., Goodwin, B., D'Andrea, A., Da Silva, M. 2001. Simulations of the SR97 Safety Assessment Case using the NUCTRAN, RSM, DSM, and PR4 codes. Ontario Power Generation Report Nuclear Waste Management Division Report 06819-REP-01200-10057-R00. Toronto, Canada.

[12] SKB. 1999. SR 97: Post-closure safety. Main Report, Volumes I and II. SKB Report 99-06.

[13] Goodwin, B.W., Melnyk, T.W., Garisto, F., Garroni, J.D., Kitson, C.I., Stroes-Gascoyne, S., Wojciechowski, L.C. 2002. Validation of Four Submodels of SYVAC3-CC4, Version SCC402. Ontario Power Generation Report Nuclear Waste Management Division Report 06819-REP-01300-10057-R00. Toronto, Canada.