Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

ACHIEVING RELIABILITY IN SIMULATION SOFTWARE

Ronald C. Van Wagenen
Charles R. Harrell, Ph.D.

PROMODEL Corporation
1875 South State Street
Suite 3400
Orem, Utah 84058, U.S.A.
(801) 223-4600

ABSTRACT

The complexity of simulation software makes it one of the most challenging software products to test and debug. Providing reliable simulation software requires that the user interface, graphic editor, statistical generators, database and memory management systems, simulation engine, output report generator, graphical animation, etc. are all working correctly. This paper provides an inside look into the challenges and issues associated with insuring the reliability of simulation software. Special emphasis is given to automated testing methods.

1 INTRODUCTION

Simulation software is becoming increasingly more complex as users demand more powerful and intelligent constructs. It is not uncommon for simulation software to contain over 500 different features and functions. The possible combinations of language features are almost infinite. Throughout the software, hundreds of equations are used to evaluate expressions of many types. Each equation must, in and of itself, be computed correctly in order to achieve accurate results. Additionally, algorithms are written for handling, not only standard logic, but also a variety of "special cases". Simulation software is very dynamic in nature, with enhancements and new language features being added continuously. Therefore it is critical that it perform correctly over time, since the model driving the program may contain any possible combination of language elements and statistical inputs.

Reliability in software is the ability of the software to perform, as documented, over a period of time. This includes the proper execution of statements or commands. If the software has logic elements, such as variables, attributes, if-then statements, etc., it is easy to see how the combinations of such functions can be endless. Reliable simulation software must be able to execute the many combinations of logic -- no matter how long a model runs and no matter how many different types of models are developed.

Totally reliable simulation software given the complexity and rapid pace of change, is only an ideal. In today's simulation technology, there is no such thing as "bug free" software. Errors in software can be measured by two criteria: probability of occurence and severity (see Figure 1). Priority 1 errors are critical and must be resolved immediately. Priority 2 errors are moderate in importance and should be resolved only after all priority 1 errors are resolved. Priority 3 errors are usually cosmetic or of minor importance. All errors must be evaluated in terms of correction time. Spending time to correct moderate or minor errors must be weighed against the time involved in implementing new features in the software. After all, the goal is to maximize the ability of a modeler to successfully complete his/her project.

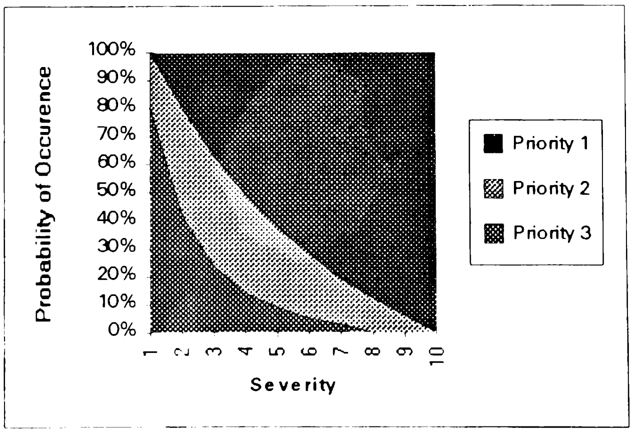

Figure 1: Error Classification

A lot of emphasis is placed on validation of simulation models. However, if the software itself is not reliable for the intended use, there is no point in validating the model. At PROMODEL Corporation, we believe that software reliability can be achieved through a combination of 1) Quality software design and coding and 2) The proper organization and execution of tests to validate the software. Since proper treatment of both concepts would exceed the size limitations of this paper, we will only discuss the latter. This includes: the function test process, comprehensive auxiliary testing, logic and output verification in each test model, and automation of testing.

2 GENERAL TESTING CONCEPTS

A logical approach to debugging simulation software, is to organize the various functions the software provides into groups of functions and test each function in the group in a similar fashion. Each function can be given a number/letter reference ID similar to a part number. Special test cases can be developed for the function to ensure that the operation of the function is not only accurate but repeatably accurate.

When a function is being designed to be incorporated into the software, the options that may be used with the function, the limitations of the function, and the valid usage of the function must be determined. This is done through written specifications. A function may have multiple options associated with its use. For example, a *zoom* function may allow zooming to any percentage or to pre-defined positions. A *write* statement may allow for writing a string value, the value of a token (such as a variable), the number of digits of the token, etc. to an external file. Test cases can be developed to test all options of a function. A set of tests or test cases is known as a test procedure.

3 FUNCTION TEST PROCESS

Once the specifications for the function have been developed, the function must pass through a series of tests we call *the function test process* (shown in Figure 2). Testing of the basic operation of the function is known as *Unit Testing*. Test cases are developed to make sure that the fundamental aspects of the functions are intact. If there are any problems, software fixes must be made before the function can be advanced to more detailed testing.

Once the function has been proven to perform when used correctly, the function must be tested when used incorrectly and under extreme conditions. Programs that crash when functions are used incorrectly are not considered user-friendly products. Warning messages and/or automatic corrections must be provided to aid the user in responding to problems. If errors still occur, they should not result in data losses or system crashes.. The testing of incorrect usage or extreme usage of the function (such as testing the maximum and minimum options) is known as *Component Testing*. This is typically the most time consuming step in the function test process. It sometimes may tie up hardware or testing resources for long periods of time. For example, when testing the minimum and maximum values that can be used for run length of a simulation, a test engineer may need to run the test model dozens of times under different conditions (with warm-up hours, without warm-up hours, with an expression representing the run hours, etc.) in order to prove that the function is reliable.

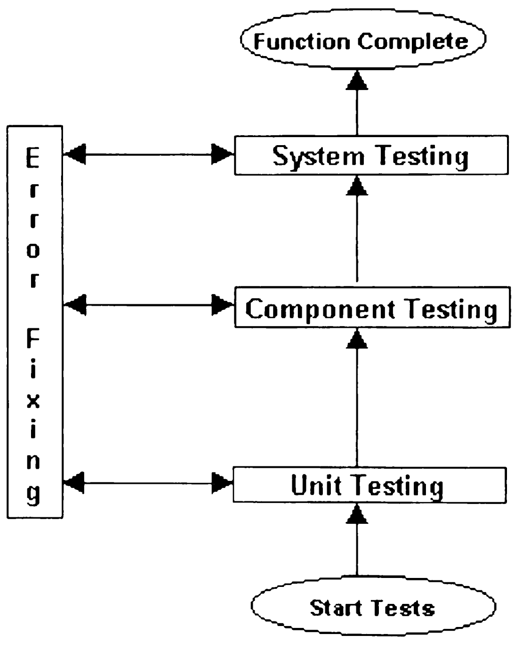

Figure 2: Function Test Process

Finally, a function may work great when used by itself but have difficulties when used with another function. The testing of multiple functions when used in combination is known as *System Testing*.

With simulation software, the best way to test the combinations of functions is through application modeling. In the development of an application model, multiple interface functions will be used to create the model, and each application model will contain a different combination of logic elements. During the model building and running, errors can surface simply because a function that was used conflicted with another function.

Once the function has successfully passed through a variety of system tests, the test procedure for the individual function is complete. A function may need to pass through the process many times in order to reach complete status.

4 AUXILIARY TESTING

When the functions of a simulation product are working well together, auxiliary tests must be performed to ensure that the product will function as a whole under a variety of circumstances. *Display Testing* procedures test the look and feel of the product to ensure that dialogs, menus, and other interface functions are consistent and balanced. *Configuration Testing* procedures test the functions of the product on a variety of hardware and software configurations to ensure that the product executes functions independent of the computer's hardware or software configuration. *Stress Testing* involves the repeated operation of test procedures in a variety of sequences and under varying memory loads to make sure the product has integrity or can withstand the stresses and demands placed upon it. *Documentation Testing* must be performed so that help screens, menus, dialogs, syntax, etc. match the manual(s) that accompany the program. *Usability Testing* experiments can also be performed to determine ease-of-use, proper documentation, and proper functional performance.

Before a simulation product can be released to the field, a select group of users can test the program using a variety of application models to make sure the interface, animation, and results are completely stable. This is known as *Beta Testing*.

5 AUTOMATED TESTING

Whenever a change is made to the source code of a simulation product, there is the potential for regression of the functionality of the software. Sometimes even the slightest changes to one feature can disable or hinder the functionality of other features in the software. In order to ensure that all of the functionality is maintained, a battery of tests can be run against the software, including running many simulation models, comparing the output results to previous runs, and testing the program interface. This can be very tedious when you consider the fact that there may easily be 10 to 30 different tests for each of the 500 or more functions. For this reason, automated software testing is extremely beneficial for performing regression testing.

There are a number of tools available which allow users to create automated tests. Such tests are written in various programming languages and are designed to test interface functions and other functions of software products. For example, an automated test can be written to load a file, such as a model, run the model, and compare the results file to the results file from a previous run.

Automation tests or scripts can be batched together and can run unattended. This is particularly useful for long model runs or long interface tests which can be run at night and not tie up computer resources during the day. Unattended testing also does not tie up personnel and can be done much more quickly than by manual methods.

Specific functions include the ability to capture menus, windows, specific regions, and text. In addition, wait states can be programmed in to re-test for a specific condition according to a time interval and to time out after a specified amount of time has elapsed. Other functions include: file existence tests, module existence tests, keystroke execution, and mouse clicks and drags. Table 1 shows which automation tools are most commonly used for the various types of testing being performed.

Model verification is a key element in automated

Table 1: Applications of Automated Testing Tools

Automated Testing Tools	**Interface Testing**	**Model Run Testing**	**Output Testing**
Window Capture Tests	√		
Text Capture Tests	√		√
Module Existence Tests	√		
File Existence Tests		√	√
File Comparison Tests			√
Mouse Clicks	√	√	
Mouse Drags	√	√	
Keys Presses	√	√	
Delays and Wait States		√	

testing. Verification in simulation means that logic elements in the simulation model execute according to specification during the run, the animation is accurate, and the results are generated correctly. Once a test model has its own logic verification algorithm, the model logic, output and animation can be verified during the automated test using the tools mentioned previously.

To create a logic verification algorithm, language elements can be used in the model to prove that the function was performed properly. For example, the following algorithm might be used to verify that a consolidation of 5 parts into 1 is occurring:

```
Begin
  5 parts enter location
  Variable1 is incremented for each part entering
          the location
  Each part is joined to batch
  Variable2 is incremented for each part leaving the
          location
  If Variable1 is not equal to 5 or Variable2 is not
          equal to 1 Then
          Write to automated log file "Logic failed"
  Else continue
End
```

Many elements, such as variables, arrays, attributes are programmed into the model with the sole purpose of verifying that standard features and statements are working correctly. In many cases, it is helpful to interface with external files, such as reading from and writing to external text files when a model passes a given test. An automated test script can be written to load the model and run it. During the run, the logic in the model will write out to the log file when a function passes or fails. At the end of the automated test, a log of test case results can be viewed to determine the status of the software function.

A second verification method used with automation is output verification. Once a set of results is proven to be credible, through manual calculations, the results file can be used as a master file which serves as the standard for subsequent runs of the model. Whenever changes are made in the software, the model can be run again, and the results of the run can be compared to the master results file for that model. There are various software tools that perform line by line comparisons of two files. Typically, such tools display a list of differences between the two files. Depending on the changes to the software, changes to the results may or may not be expected. If so, the changes are usually very minor. Such changes will show up in the file comparison analysis.

Still another important verification method which can be automated is interface verification. This means that the menus, modules, and keystroke functions of a program are tested. Automation tools such as window capturing, menu capturing, module existence testing, etc. can be used to verify that the interface of the product is intact. In fact, once tests have been developed for the various elements of the interface, a special sequence of keystrokes and menu clicks can be developed to represent an actual user of the product. This sequence can also be developed into a random sequence, which is especially useful for stress testing.

6 SUMMARY

Achieving complete reliability in simulation software is no small task. There are three critical levels of testing that must be performed for each function: *Unit Testing*, *Component Testing*, and *System Testing*. Then the product must be tested as a whole on different types of configuration, under varying stress loads, etc. Once the tests have been developed, automation tools can be used to take over the responsibilities of regression testing. This enables software vendors to "manufacture" software. In other words, if a special change is needed for a simulation user, the change can be made, automated regression testing can be performed (usually within a day), and the software can be sent out with confidence and reliability intact.

REFERENCES

Beizer, Boris 1984. Software System Testing and Quality Assurance, New York, New York, Van Nostrand Reinhold

Besterfield, Dale H., 1986. Quality Control, 2nd. Ed., Englewood Cliffs, New Jersey, Prentice-Hall

Hetzel, Bill 1988. *The Complete Guide to Software Testing*, 2nd Ed., Wellesley, MA, QED Information Sciences, Inc.

Kelton, W. David and Averill M. Law 1991. *Simulation Modeling and Analysis*, New York, New York, McGraw-Hill, Inc.

Rabin, Steve 1993. Software Testing: Concepts, Tools, & Techniques, Software Development, November, pp. 63-71

Sargent, Robert G. 1992. Validation and Verification of Simulation Models, *Proceedings of the 1992 Winter Simulation Conference*, Arlington, Virginia, pp. 104 - 114

Software Quality Automation Inc., 1993, *SQA Robot User Guide*, Woburn, MA

AUTHOR BIOGRAPHIES

RONALD C. VAN WAGENEN is Testing Manager and Applications Engineer at PROMODEL Corporation. He received a B.S. in Manufacturing Engineering Technology from Brigham Young University. He has worked for IBM in software testing and development, J.I. Case in manufacturing engineering, and has worked for PROMODEL for the past 4 years. He is a member of **IIE** and **SME**.

CHARLES R. HARRELL is Chairman and founder of PROMODEL Corporation and an Assistant Professor of Manufacturing Engineering at Brigham Young University. Charles received his B.S. in Manufacturing Engineering Technology from Brigham Young University, M.S. in Industrial Engineering from the University of Utah, and Ph.D. in Manufacturing Engineering from the Technical University of Denmark. He has worked as a simulation analyst and systems designer at Ford Motor Company and Eaton Kenway. Charles headed the design and development of PROMODEL Corporation's Windows-based simulation products: ProModel, MedModel and ServiceModel. He is a member of **IIE** and **SME**.