Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

VERIFICATION AND VALIDATION OF OBJECT-ORIENTED ARTIFACTS THROUGHOUT THE SIMULATION MODEL DEVELOPMENT LIFE CYCLE

John T. Carr, III

Naval Surface Warfare Center
Dahlgren Division, Code T12
17320 Dahlgren Road
Dahlgren, VA 22448-5100, U.S.A.

Osman Balci

Department of Computer Science
660 McBryde Hall, MC 0106
Virginia Tech
Blacksburg, VA 24061, U.S.A.

ABSTRACT

The purpose of this paper is to present a series of questions (or indicators) for assessing the verity and validity of the artifacts produced during the entire object-oriented simulation model development life cycle. Using modern object-oriented development processes, artifacts developed in one phase flow seamlessly from those of the previous phase. This provides forward and backward traceability between artifacts. This inherent backward traceability has been exploited by tracing defects in artifacts back to their defective ancestral artifacts. Questions are then phrased such that when answered in the negative indicate the presence of defects. Use of the Evaluation Environment software tool facilitates the integration of the answers to the assessment questions and enables an overall evaluation. The collection of questions can be useful for the verification and validation of artifacts in any object-oriented simulation model development.

1 INTRODUCTION

A simulation model development life cycle consists of processes and products. “Process” refers to a series of activities conducted to create a life-cycle product, such as engineering the model requirements, designing the model, or generating the model code. “Product” refers to a different characterization of the model during its development life cycle, such as the model requirements specification, model design specification, or executable model.

A model characterization is transformed from one product form (e.g., design) into another (e.g., code) by carrying out a process (e.g., programming) during the development life cycle. *Verification* deals with transformational accuracy. *Validation* deals with representational or behavioral accuracy. Since a model, by definition, is an abstraction of what it represents, perfect accuracy cannot be expected. Therefore, the sufficiency of accuracy is judged with respect to the model intended uses and project objectives.

One of the principles of verification and validation (V&V) dictates that defects should be detected as early as possible in the development life cycle (Balci 1997). However, this is not always achievable and we encounter many defects in later stages of the life cycle. Identifying the origin of a defect is very challenging especially if the defect is found in later stages of the development.

Generally, two approaches are commonly used for simulation model development: procedural and object-oriented. In *procedural development*, the representations of model requirements, high-level model design, detailed model design, and executable model are created under different conceptual frameworks and significantly differ from each other. The conceptually different model representations, throughout the development life cycle, require the modeler to make sophisticated transformations from one conceptual framework to another. Such transformations are known to be error-prone and require much more V&V.

In *object-oriented development*, on the other hand, the same conceptual framework is used throughout the entire development life cycle starting with problem domain analysis and culminating with the modeling and simulation (M&S) application. Transformation of object-oriented model representations from one form into another is greatly facilitated since the same concepts and fundamentals are used in all model representations.

In using the object-oriented development approach, some artifacts are commonly created such as use cases, use case diagrams, sequence diagrams, and class diagrams. Modern object-oriented development processes provide traceability between these artifacts over the span of the development. Since V&V is not a stage, but a continuous activity, V&V of the object-oriented artifacts is carried out throughout the entire development life cycle.

One of the key advantages of the object-oriented approach is *traceability*. Traceability between object-oriented artifacts results in traceability of defects. When a defect is identified in a model representation form (i.e., an artifact), it can be traced back, through the sequence of artifacts, to the origin of the defect in an earlier artifact. Thus, the artifact in which the defect is first introduced is identified. Removal of the defect results in the correction of the artifacts derived from the corrected artifact. However, stopping at this point would be premature since it misses the opportunity to make improvements in the model development process by attempting to identify the reason(s) why the artifact was defective in the first place (Texel and Williams 1997). Deming (1986) indicates that 90% of all product defects can be traced to defective processes.

Software development consists of *macroprocesses* and *microprocesses*. Macroprocesses, such as those comprising the Unified Method (Booch, Rumbaugh, and Jacobson 1999; Jacobson, Booch, and Rumbaugh 1999; Rumbaugh, Jacobson, and Booch 1999), have evolved to the point that they are mature, easy to follow, and are unlikely sources for defects. Therefore, when defects occur using a macroprocess we must examine the microprocesses, which describe the specific steps and factors used to develop an artifact. We can question the reason why a defect occurred in terms related to the microprocesses by tracing the defect back to the artifact in which it was originally introduced.

Software metrics (de Champeaux 1997) allow one to quantify various properties of object-oriented artifacts. However, their interpretation requires a history of their use with a given process by a given organization. Currently, this is usually lacking. In addition, independent verification, validation, and accreditation (VV&A) agents must assess software artifacts developed by a variety of organizations using a variety of processes. Consequently, at the present time, software metrics usually do not provide the VV&A agents with a consistent process for assessing artifacts. As an alternative, we propose the use of assessment questions (or indicators).

The remainder of this paper is organized as follows. The assessment questions are presented in Section 2. Section 3 discusses the evaluation of answers to the assessment questions using the Evaluation Environment™ software tool. Completeness of the assessment questions is discussed in Section 4. Conclusions are given in Section 5.

2 V&V USING ARTIFACT ASSESSMENT QUESTIONS

Questions (also called indicators) are developed in terms of the microprocesses for use cases, use case diagrams, sequence diagrams, and class diagrams. Given an artifact for a simulation model under development, the questions can be applied to verify and validate the artifact before it is used to develop subsequent artifacts.

2.1 V&V of Use Cases

A use case describes a single use of a system, including alternatives and exceptions, by one or more actors. Among other formats suitable for documenting use cases include those of Texel and Williams (1997, pp. 46-49), Larman (1998, pp. 55-63), or the formal syntax presented by de Champeaux (1997, pp. 86-87). Through our inspection of artifacts it has been determined that defects are more likely to occur when different formats are used within the same project. Therefore, the first set of assessment questions for use cases is:

1. Is the use case diagram drawn using a standard template?
2. Do actors and use cases follow a standard naming convention and format?

All of the previously mentioned formats contain a form of user action and system response, which are repeated over and over. The user action is performed by an actor. The format clearly indicates that the actor should not be part of the system. An example of a defect of this type is using a database as an actor. A database is not a user of a system. It is part of a system. The result is defective sequence diagrams that have no clear indication of the start of the use case. This leads to four additional assessment questions for use cases:

3. Are the actors external to the system?
4. Are the actors external to the use case boundary?
5. Does an action by a user start each use case?
6. Is the start of each use case unambiguous?

The previous use case questions deal with the issue of structural correctness of use cases. The following questions assess the ability of a use case to capture a subset of the system requirements:

7. Does the use case make sense?
8. Does the use case accurately represent the behavior specified in the requirements?
9. Does the use case cover all paths including decisions, alternates, and exceptions?
10. Are the preconditions correct?
11. Does the use case produce useful and appropriate results, i.e., are the post conditions correct?
12. Are the requirements captured by the use case specified?
13. Should similar use cases be combined into a single use case?
14. Should the use case and associated requirements be divided into several requirements and use cases?

15. Is each functional requirement associated with at least one use case?
16. Are use cases sharing one or more functional requirements consistent?
17. Can the use case be tested?

2.2 V&V of Use Case Diagrams

It is important that each use case diagram be consistent with the corresponding use case. This consistency can be assessed through the application of the following questions:

1. Is there a use case diagram for each use case?
2. Are all use case diagrams drawn using the same, preferably the UML diagramming notation?
3. Is each actor represented in the use case diagrams in which it is involved?
4. Should similar use case diagrams be combined using use case associations such as extension of a use case and the use of a use case by another?

2.3 V&V of Sequence Diagrams

The following questions are created from the required traceability between the use cases and sequence diagrams.

1. Are the sequence diagrams drawn using a consistent, preferably UML, notation?
2. Is there a use case from which the sequence diagram derives?
3. Is there an actor that initiates each sequence diagram the same one that initiates the use case from which it is derived?
4. Are all nouns, noun phrases, and verbs that imply creation represented as objects?
5. Does the diagram have a meaningful termination?
6. Do all of the objects present in the sequence diagram have associated classes in the design class diagram?
7. Do the diagrams include alternatives and exceptions?
8. Is there at least one sequence diagram for each use case?

2.4 V&V of Class Diagrams

Aside from the source code itself, the class diagram is probably the most important artifact related to the maintainability of object-oriented software. The design class diagram contains the pattern for what is implemented in code. The constraints imposed on relationships between classes as specified in the class diagram determine how the software can be modified, extended, and reused. Where possible class diagrams should be based on design patterns (Gamma *et al.* 1995). These are patterns that demonstrate proven relationships between classes that address specific design problems.

1. Is the class diagram drawn using UML?
2. Do all sequence diagram objects have associated classes in the class diagrams?
3. Are design patterns used to create associations for common relationships?
4. Are there classes other than containers for which there are no corresponding objects in the class diagrams?
5. Are classes present not traceable to the requirements or use cases?
6. Are roles identified?
7. Are redundant classes present?
8. Are multiplicities shown and correct?
9. Are containment and aggregation used properly in the diagram?
10. Are container, boundary, control, association, service, and creator classes that do not correspond to objects in the problem domain avoided?

The inheritance class model can be assessed by answering the following questions:

1. Is each derived class "a kind of" its base class?
2. Does each derived class implement one or more base class operations?
3. Can a derived class object be used wherever an instance of the base class is used?
4. When multiple inheritance is used, are all ambiguities addressed?

The design class diagram can be assessed by answering the following questions:

1. Is the design class diagram consistent with the conceptual model?
2. Are roles identified?
3. Are multiplicities specified?
4. Are containment and aggregation present in the diagram?
5. Are container, boundary, control, association, service, and creator classes that do not correspond to objects in the problem domain present?
6. Do the classes have low coupling?
7. Do the classes have high cohesion?
8. Are attributes mistaken for classes?

3 EVALUATION OF ANSWERS TO THE ASSESSMENT QUESTIONS

The Evaluation Environment™ (EE) (Orca 1999) software tool can be used to facilitate subject matter expert (SME) evaluation using the assessment questions. This tool lets SMEs evaluate the importance of a “yes” or “no” answer to each assessment question and then aggregates the individual assessments to obtain an overall assessment. This allows consideration of the importance of a “no” answer to a question to the overall assessment of the completeness, correctness, clearness, and consistency for an artifact.

In the case of multiple inheritance, some SMEs believe that multiple inheritance should never be used while other SMEs disagree. When assessing the use of multiple inheritance in an artifact, the EE tool can provide for consideration of these varied opinions and corresponding evaluations in a quantitative manner as opposed to a subjective approach.

Different SME judgments are integrated in the EE tool by using *relative criticality weighting* of SMEs. Weights are used to express an SME’s level of influence in the integration of all SME judgments. A *weight* is a fractional value between zero and one. The weights of the SMEs being compared must sum to one.

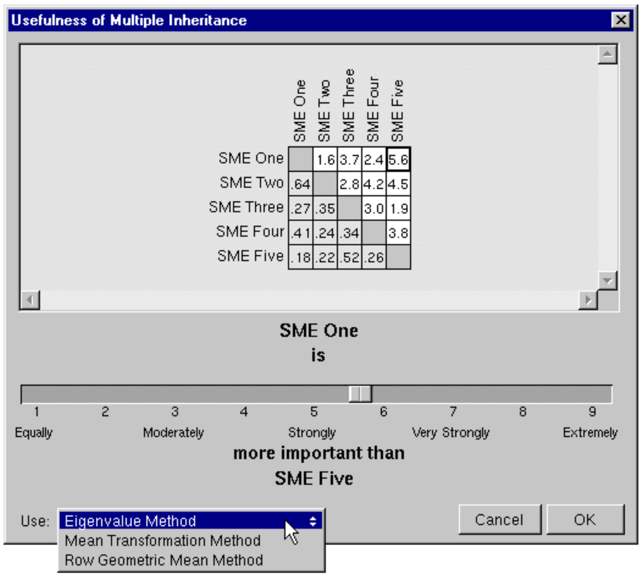
Given a list of *n* SMEs, it is very difficult to come up with numerical weights especially when *n* > 5. To facilitate the relative criticality weighting, the mathematical approach called Analytic Hierarchy Process (AHP) is commonly used in the multicriteria decision making field (Saaty 1994). AHP enables pairwise comparisons of importance between the SMEs and computes the weights based on the pairwise comparison values using methods such as Eigenvalue method, Mean Transformation method, and Row Geometric Mean method.

Figure 1 depicts the relative criticality weighting of five SMEs, by using the EE tool, in the integration of their judgments on the usefulness of multiple inheritance for a given artifact. Clicking on a matrix cell displays the pairwise comparison of the corresponding two SMEs. Using the sliding bar, a judgment of relative importance is specified. Any of the three techniques can be selected from the pull-down menu for the EE tool to calculate the numerical weights by using AHP. Selecting the Eigenvalue method for the pairwise comparisons shown in Figure 1, we obtain the numerical weights shown in Figure 2, which is created as a Kiviat graph by the EE tool.

4 COMPLETENESS OF THE ASSESSMENT QUESTIONS

The assessment questions above were developed and tested through applications to the following software systems:

1. ICU2: Satellite coverage software (Carr 1988). (The use case and use case diagram assessment questions were not applied to ICU2.)
2. Ranger: Satellite tracking data simulator (Carr 1994b)


Usefulness of Multiple Inheritance

	SME One	SME Two	SME Three	SME Four	SME Five
SME One		1.6	3.7	2.4	5.6
SME Two	.64		2.8	4.2	4.5
SME Three	.27	.35		3.0	1.9
SME Four	.41	.24	.34		3.8
SME Five	.18	.22	.52	.26	

SME One
is

1 Equally · 2 · 3 Moderately · 4 · 5 Strongly · 6 · 7 Very Strongly · 8 · 9 Extremely

more important than
SME Five

Use: Eigenvalue Method
Mean Transformation Method
Row Geometric Mean Method

Cancel OK

Figure 1: Relative Criticality Weighting of SMEs

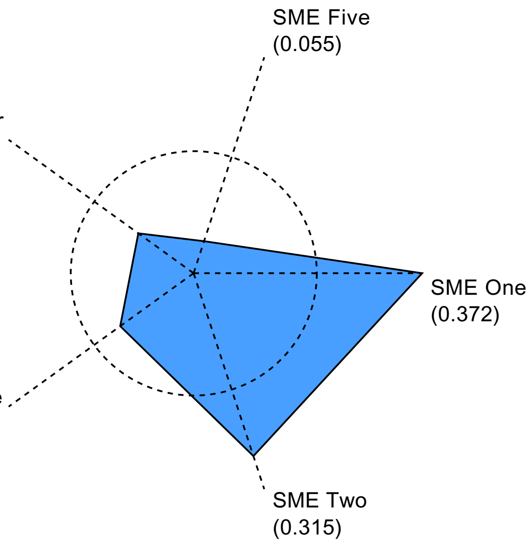


Figure 2: Kiviat Graph Illustrating the Criticality Weighting of SMEs

3. Alert: Tracking station scheduler program (Carr 1994a)
4. ASTER: Advanced satellite trajectory estimation routines (Carr 1992)
5. SWIPE: Simple windows programming environment (a Motif-based class library) (Carr 1993)
6. HFSS: High fidelity system simulation (missile defense simulation)
7. LIDS: Lead system integrator integration distributed simulation (missile defense simulation)

Thus the assessment questions have been demonstrated to be suitable for assessment of problems in the scientific problem domain. However, since the aspects of the artifacts assessed by the questions are generic properties desirable in object-oriented artifacts, they should also be applicable in other application domains.

UML defines eight types of diagrams: use case, class, sequence, activity, statechart, collaboration, component, and deployment (Binder 2000). The software projects used in developing the assessment questions only produced use case, class, sequence, activity, and state transition diagrams. Even all of these projects did not produce all of the diagrams mentioned. As a result, assessment questions for state transition diagrams and activity diagrams have not been fully developed. No questions have been developed for collaboration, component, and deployment diagrams.

5 CONCLUSIONS

The verity and validity of object-oriented artifacts can be assessed through the application of a series of assessment questions. These questions assess correctness, completeness, consistency, clarity, and testability of each artifact. Using these questions, defects can be identified early in the development life cycle and the VV&A agent can provide corrective feedback to the M&S application developer before the defect is propagated through the remainder of the development life cycle. Use of the EE software tool facilitates the overall assessment of artifacts based on SME responses to the assessment questions.

REFERENCES

Balci, O. 1997. Principles of simulation model validation, verification, and testing. *Transactions of the Society for Computer Simulation International* 14 (1): 3-12.

Binder, R.V. 2000. *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley, Reading, MA.

Booch, G., J. Rumbaugh, and I. Jacobson. 1999. *The unified modeling language user guide*. Addison-Wesley, Reading, MA.

Carr, J.T. 1988. *ICU2: satellite coverage software.* Naval Surface Warfare Center, Dahlgren, VA.

Carr, J.T. 1992. *ASTER: advanced satellite trajectory estimation routines*. Naval Surface Warfare Center, Dahlgren, VA.

Carr, J.T. 1993. *SWIPE: simple windows programming environment.* Naval Surface Warfare Center, Dahlgren, VA.

Carr, J.T. 1994a. *Alert: tracking station scheduler program.* Naval Surface Warfare Center, Dahlgren, VA.

Carr, J.T. 1994b. *Ranger: satellite tracking data simulator.* Naval Surface Warfare Center, Dahlgren, VA.

de Champeaux, D. 1997. *Object-oriented development process and metrics*. Prentice-Hall, Upper Saddle River, NJ.

Deming, W.E. 1986. *Out of the crisis.* Center for Advanced Engineering, Massachusetts Institute of Technology, Cambridge, MA.

Gamma, E., R. Helm, R. Johnson, and J.M. Vlissides. 1995. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, Reading, MA.

Jacobson, I., G. Booch, and J. Rumbaugh. 1999. *The unified software development process*. Addison-Wesley, Reading, MA.

Larman, C. 1998. *Applying UML and patterns,* Prentice-Hall, Englewood Cliffs, NJ.

Lorenz, M. and J. Kidd. 1994. *Object oriented software metrics.* Prentice-Hall, Upper Saddle River, NJ.

Orca 1999. *Evaluation environment user's guide*. Orca Computer, Blacksburg, VA.

Rumbaugh, J., I. Jacobson, and G. Booch. 1999. *The unified modeling language reference manual*. Addison-Wesley, Reading, MA.

Saaty, T.L. 1994. *Fundamentals of decision making and priority theory with the analytic hierarchy process*. RWS Publications, Pittsburgh, PA.

Texel, P.P. and C.B. Williams. 1997. *Use cases combined with Booch/OMT/UML: process and products*. Prentice-Hall Canada, Toronto, Ontario, Canada.

AUTHOR BIOGRAPHIES

JOHN T. CARR, III is a Scientist at the Naval Surface Warfare Center, Dahlgren Division. He received his M.S. degree from Virginia Tech in 1985. From 1973 to 1995 Mr. Carr worked on the research and development of space systems. His areas of expertise include space system simulation development, formulation of satellite orbit determination algorithms, satellite force model development, space system visualization, object oriented software development, and Modeling and Simulation Verification, Validation, and Accreditation. Since 1996 Mr. Carr has worked on missile defense systems. Currently, Mr. Carr is lead for Simulation Verification, Validation, and Accreditation within the National Missile Defense Verification and Validation Program at the Naval Surface Warfare Center. His e-mail address is `<CarrJT@nswc.navy.mil>`.

OSMAN BALCI is Professor of Computer Science at Virginia Tech and President of Orca Computer, Inc. He received his Ph.D. degree from Syracuse University in 1981. Dr. Balci is the Editor-in-Chief of two international journals: *Annals of Software Engineering* and *World Wide Web*; Verification, Validation and Accreditation (VV&A) Area Editor of *ACM Transactions on Modeling and Computer Simulation*; and Modeling and Simulation

(M&S) Category Editor of *ACM Computing Reviews*. He serves as a member of the Winter Simulation Conference Board of Directors representing the Society for Computer Simulation. Most of Dr. Balci's research has been funded by DoD since 1983. Recently, he has provided technical services for the National Missile Defense (NMD) program in the areas of NMD system design M&S VV&A and NMD system design IV&V. His current research interests center on VV&A, IV&V, certification, quality assurance, and credibility assessment of (a) M&S applications, (b) software systems, and (c) complex software / hardware / humanware system designs. His e-mail and web addresses are `<balci@vt.edu>` and `<http://manta.cs.vt.edu/balci>`