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

SOFTWARE ENGINEERING BEST PRACTICES APPLIED TO THE MODELING PROCESS

David H. Withers

Dell Computer Corporation
One Dell Way 6370
Round Rock, TX 78682, U.S.A.

ABSTRACT

We present a mapping of Best Practices from the field of software engineering to the practice of discrete event simulation model construction. There are obvious parallels between the two activities. We therefore hypothesize there should be opportunities to improve the model construction process by taking advantage of these parallels. This research extends the prior work (Withers 1993) that provided a structured definition of the modeling process.

1 INTRODUCTION

Both activities (software engineering and discrete event simulation modeling) require a specification of the desired result, design of the system to be built or modeled, development of a project plan, coding and testing of software components, and verification of the result against the original specification. Both activities also suffer from a history of missed schedules, broken promises, and cost overruns. Documented evidence of failures is more prevalent in the software engineering community than in the modeling community. However, indirect evidence of the need for continuing improvement of the modeling process has been presented at each Winter Simulation Conference in recent history. The Program Chairs have routinely included a full-length invited tutorial on insuring success in modeling. Distinguished, successful practitioners have provided a lineage of papers, many with humorous approaches, and solid guidance on things to be sure to do or not do to raise the probability of success in modeling projects (Sadowski 1999, Robinson 1995 and Musselman 1994). It is interesting to note these landmark papers have usually not included an emphasis on the software engineering aspects of the modeling process except to point out the need for a specification and for testing throughout the project.

There has also been a continuing emphasis on verification, validation, and accreditation as part of the modeling process (Arthur 1996, Balci 1997). There is a strong parallel between these recommendations and the testing recommendations from the field of software engineering.

The modeling process has been the subject of numerous research and practitioner reports. Textbooks usually include a section or chapter on the process one should follow to improve the probability of success in modeling (Pritsker 1999, Law and Kelton 1999). Research reports have proposed formalisms to structure the process (Withers 1993) and practitioner reports have provided guidance based upon observed successes, see for example Hewitt (1999).

An area of research and application that provides a direct overlap between the modeling and software engineering practices is the maturing of design, analysis, and development of object oriented software components. This technology has shown promise in improving both modeling and software engineering (Joines 1995). There have also been reports on efforts to develop special purpose tools for modeling (Zeigler 1990, Zeigler 1993) and for statistical analysis of input and output data (Law 1999).

One might argue that modern simulation modelware relieves the modeler of programming; therefore there is no need to improve the part of the modeling process related to programming. We argue that the new modelware tools raise the level of programming so that certain skills (e.g., how to program an event list) are no longer required, but the logical constructs and semantics of programming are still required. Some tools relieve us of the burden of formatting, but none relieve us of the burden of translation of real system logic into simulation logic.

2 BACKGROUND ON SOFTWARE ENGINEERING BEST PRACTICES

This section briefly reviews the history of Best Practices in the software engineering area. The U.S. Department of Defense has a long-standing interest in the success of software engineering projects and has nurtured several

initiatives to both measure and improve the process. Significant activities in this context were sponsorship of the Software Engineering Institute and formation of the Airlie Software Council.

The Software Engineering Institute at Carnegie Mellon University is a federally funded research and development center established in 1984 by the Department of Defense to address the improvement of software engineering technology. Their web site, <http://www.sei.cmu/sei-home.html>, is an excellent source of information on best practices.

The Airlie Software Council was sponsored by the Department of Defense (DOD) to improve success on large complex software projects. The council produced the original list of 9 essential Best Practices in 1994-95 (noted below as "Original 9) (Brown, 1999). The original list has been modified and extended to the current list of 16. We will provide a brief statement of each of these, map them to the simulation modeling process as defined earlier (Withers, 1993), and then suggest how to apply them in the simulation modeling context.

In this context a Best Practice is defined as a management or technical practice that has consistently demonstrated to improve one or more of:

Productivity
Cost
Schedule
Quality
User Satisfaction
Predictability of Cost and Schedule (McGrath 1998)

McGrath (1998) presented a discussion of the current list. His discussion is the primary basis for our summary remarks below. The Software Program Managers Network (SPMN) has categorized the Best Practices into the three major categories below (SPMN 1999). In the next section we provide a short discussion for each Best Practice in software engineering context.

3 SOFTWARE ENGINEERING BEST PRACTICES

Here we summarize the best practices from the perspective of software engineering. They are categorized using the Software Program Managers Network structure and are annotated to show which of them were part of the original set.

3.1 PROJECT INTEGRITY

This category includes six (6) best practices related to project management. The discussion in this section is targeted to software engineering and/or project management in general. Later we will relate these to modeling projects.

1. **Formal Risk Management** (Original 9). This is the number one Best Practice according to the Airlie Software Council (Lister 98). The risks are present whether or not they are acknowledged. The Best Practice is a formal process for identifying, addressing, and mitigating risk items before they negatively impact program quality, schedule, and cost. Risks are identified and documented along with a probability of occurrence and exposure. For direct risks, actions are taken to mitigate, avoid, or transfer the risk, and documented. For indirect risks, contingency plans are agreed to and documented.
2. **Estimate cost and schedule empirically.** Cost and schedule estimates should be based upon empirical, historical data, early size estimation, and tracking of project status through the use of captured-result metrics. Inaccurate cost and scheduling estimates result in cost overruns. Schedules slip because software specifications constantly change, deadlines are often arbitrary and impossible to achieve, and numerous events are interdependent. Metrics such as cost, effort, schedule, successful activity completion and defects provide the means to track and manage the project.
3. **Metric-based Scheduling and Management** (Original 9). Project planning and management should be based upon accumulated evidence over multiple projects. Planning should be at a measurable level of detail:
   Each activity in the project plan should reflect a Binary Quality Gate at inch-pebble level (Original 9). The project plan should be decomposed to activities where each activity has a short span (no longer than a day) (no longer than an inch and no heavier than a pebble!) and an easily measured completion – it's either done or it's not done (binary gate). The concept of percent completion at the activity level is viewed as absurd.
   Program wide visibility of progress vs. plan (Original 9). Chances of effective risk management and program success are improved when the entire project is visible to all participants. Early warning of potential completion problems saves all participants money, time, embarrassment, and operational impact.
   A core set of metrics which fall into these four categories should be used by every project:

   Early warnings of potential problems.
   Product quality – See section 5. below.

Effectiveness of process improvement, the ability to determine whether process changes are working better on the next project.
Actual costs, labor and materials converted into currency.

4. **Track Earned Value** as the project progresses. Earned value calculations are based upon a binary assessment of each activity in the project plan. A completed activity scores 1, all other activities are 0. There is no partial credit for in-process activities. At any point the total completions divided by the number of activities in the plan is a very accurate measure of progress. As noted above, project plans should be built at the inch-pebble detail so there are no long-running activities that cannot be accurately measured and tracked.
5. **Defect Tracking** against quality targets (Original 9) provides a measure of progress against objectives for deliverable quality. Every defect is recorded, beginning with the inspection of the specifications and continuing through all defect identification activities. There should be no such thing as a private defect – one that is detected and removed without being recorded. Simply tracking defects detected and defects closed over time provides an excellent visual display of the quality maturity of the project. Once discoveries level off and closure approaches discovery, the project becomes a candidate for deployment.
6. **Treat People** as the most important resource (Original 9). This Best Practice was originally known as People-aware Management Accountability. The most important determinants of project success may be the quality, experience, and motivation of the people working on it.

3.2 CONSTRUCTION INTEGRITY

This section describes software engineering best practices for the design and development activities.

7. **Configuration Management** (Original 9). Configuration Management is an integrated process for identifying, documenting, monitoring, evaluating, approving, and controlling all changes to project information shared by more than one individual.
8. **Manage and Trace Requirements.** Requirements need to be complete, consistent, and testable and must meet the user's needs with low volatility and 2-way traceability from design through testing. This activity was cited as the most important best practice by some researchers (e.g., McGrath 98). This Best Practice includes the use of structured methods for users to define system requirements (e.g., a user interface prototype), formal methods to find consistency errors, control of volatility, and structured peer reviews. Requirements should include operational scenarios that characterize real-world operations. They should reflect both nominal and stress conditions. Engineers who will build the system should be involved in the requirements specification process.
9. **System-based Software Design.** The practice introduces systems analysts/programmers into the project during early design to insure the project can be technically developed. The system design is subjected to an inspection before it is placed under configuration management.
10. **Ensure data and database interoperability** provides a focus on the operability of the system in context with its environment. There does not appear to be a direct correlation for modeling.
11. **Define and Control Interfaces** (Original 9). Interfaces to other systems should be agreed-upon, coded, and then maintained as a baseline. A date should be set, after which no further changes will be accepted There does not appear to be a direct correlation for modeling unless the model is an embedded system.
12. **Design Twice and Code Once** is a parallel to the old carpenter's axiom to measure twice and cut once. Avoid the rush to coding and insure the design really matches what the customer needs.
13. **Assess reuse risks and costs**. The architecture should plan for re-use starting with project planning artifacts. Implementation of this best practice is so difficult that we do not believe it will have effective applicability to the modeling process so it will not be included in the section 5.

3.3 PRODUCT STABILITY and INTEGRITY

This third section provides information on best practices for insuring the quality of the product.

14. **Formal Inspections** of requirements and design (Original 9). This practice has consistently shown to reduce rework by 30%, (Cook 1998). Michael Fagan published the original definitions for effective reviews and inspections (Fagan 1986). More defects can be removed by inspection than by any other testing process. The defects can be removed earlier in the process when they are easier and much less expensive to correct. Inspections can find defects that would not be found during testing, such as special cases when an algorithm will produce incorrect results. To be

successful, the project needs to create an environment where defect detection is encouraged, rather than reporting how few defects exist. Excellent references with further information on how to conduct inspections and reviews include Gilb (1993) or Wheeler (1996).

15. **Manage Testing as a Continuous Process.** Testing should begin when the project begins and continue through all deliverables. Testing is not limited to executable code, but can be applied to documents, prototypes, designs, and even to test plans.
16. **Smoke Test Frequently**. Smoke testing is targeted at proving the accuracy of new functions. Testing to qualify new components proceeds only after regression testing of the prior capabilities. Smoke testing cannot be performed by the development engineer, but must be accomplished by an independent organization. Results of smoke testing should be published to the entire team and all defects should be logged and tracked.

4 MAPPING SOFTWARE ENGINEERING BEST PRACTICES TO THE DISCRETE EVENT SIMULATION PROCESS

In this section we present a suggested translation of the Best Practices described above from the field of software engineering into the practice of discrete event simulation modelling. We suggest where each Best Practice should be applied to the discrete event simulation modeling process. Our motivation is to transfer the accumulated knowledge from one domain to another. We argue that the modeling process consists of many of the same steps as development of a software solution. Therefore the accumulated process knowledge as captured in the Best Practices should be transferable.

The first two columns in Figure 1 are taken directly from the structured model of the modeling process (Withers 1993). The remaining columns identify where each of the Best Practices may be applied. Details on how they may be applied are in the next section.

5 APPLICATION OF BEST PRACTICES

In this section we describe how the above best practices may be applied to the discrete event simulation process. We begin with a focus on project management.

1. **Formal Risk Management** is applied throughout the project. For modeling projects risks usually include access to historical data, access to documentation for existing and planned processes, and lack of management understanding of the modeling process and sophisticated statistical analyses required. As soon as a risk is identified it should be logged along with an assessment of the impact and the probability of occurrence. A consistent measure for impact is needed. Obviously cost is the best metric, but it is usually hard to precisely quantify each risk. We recommend a scale (e.g., 1 to 5). Similarly, the probability of occurrence is hard to quantify precisely, so we recommend using 5%, 50%, and 95% representing unlikely, may occur, and likely. The product of impact and probability of occurrence provides a ranking so one can easily produce a "top 3" or "top 10" list of risks. A spreadsheet works nicely to track risks.

   Example: Data defining service times for the new equipment may not be available in time to use as input to the model for Phase I capacity planning. The impact is: The confidence bounds for the output will increase. This is a medium (2) impact. The probability of occurrence is 50%. The ranking number is 100 (=50 x 2).

   The top n risks need to be mitigated by reducing the impact, the probability of occurrence, or both. For example, if something is likely to occur and has a high impact, the prudent project manager will plan for it and include appropriate activities in the project plan.
2. **Estimate Cost and Schedule Empirically** is applied when the plan is constructed. Some measure of project complexity should be used as a basis. For simulation projects one measure of complexity is the number of business process owners. Another is the number of different kinds of equipment (mills, forklifts, computers, etc.) that will need to be included in the model. Obviously, keeping a log of effort, duration, and complexity for prior projects will be a great asset in estimating the next project. If a history of prior projects is not available, we recommend estimating each of the activities found in the lowest level of decomposition for the modeling process (Withers 1993). Assigning an estimate to each process activity will at least provide an estimate of the complete process and while some estimates will be low and others high, the project total has a better chance of being in bounds. If the model is to be built iteratively, the number of iterations should be planned in advance. Our experience is that three iterations of Produce Conceptual Model and Produce Model are usually sufficient. The customer needs to agree in advance on the number of iterations, else the project will never end. The empirical estimate can be compared to the project plan (best practices 3 and 4 below.)

Structured Modeling Process Level 1	Structured Modeling Process Level 2	Formal Risk Management	Estimate Cost and Schedule Empirically	Metric-based Scheduling and Management	Track Earned Value	Defect Tracking	Treat People	Configuration Management	Manage and Trace Requirements	System-based Software Design	Design Twice and Code Once	Formal Inspections	Manage Testing as a Continuous Process	Smoke Test Frequently
1. UNDERSTAND CUSTOMER and SYSTEM	1.1 Define Modeling Team						■							
	1.2 Define Problem to be Solved	■					■							
	1.3 Determine Customer Constraints and Develop Plan	■	■	■	■	■	■	■	■	■		■		
2. PRODUCE CONCEPTUAL MODEL	2.1 Understand System	■			■		■		■					
	2.2 Develop/Update Conceptual Model	■			■	■	■	■	■		■			
	2.3 Develop Data Requirements	■			■	■	■	■	■		■			
	2.4 Validate Conceptual Model and Data Requirements				■	■	■	■	■			■		
3. PRODUCE MODEL	3.1 Determine Class of Model	■			■		■							
	3.2 Develop Model, Data, and Documentation	■			■	■	■	■	■		■		■	■
	3.3 Verify Model to Specifications				■	■	■	■	■					
	3.4 Validate Model to Requirements				■	■	■	■	■			■		
4. USE MODEL	4.1 Design Experiments	■			■		■							
	4.2 Execute Experiments	■			■	■	■							
	4.3 Validate Results				■	■	■							
	4.4 Make Decision on System Change	■			■		■							
5. ASSESS MODEL USE	5.1 Run Operation						■							
	5.2 Assess Operating Environment and Performance	■			■		■							
	5.3 Operate Model	■			■	■	■							
	5.4 Compare Performance Measures Model:Operations	■			■	■	■							
	5.5 Assess Documented Use of Model				■		■							

Figure 1. Mapping of Software Engineering Best Practices to Structured Modeling Process

3. **Metric-based Scheduling and Management and** 4**. Tracking Earned Value** are connected. The project plan should be built so that activities are very short (hours or a day) and can easily be assessed as either complete or incomplete. Progress is measured easily as the ratio of the number of complete activities to the total number of activities in the project plan. Project managers should avoid estimates of percent completion at the activity level.

5. **Defect Tracking** begins with inspection of the project plan and continues throughout most of the development and deployment activities. Anytime a defect is found it is logged. When a defect is fixed, that event is logged. A simple plot of cumulative defects found vs. time will provide a quick assessment of residual defects by looking for the discovery rate to level off. A plot of defects remaining (found minus fixed) vs. time provides an assessment of the readiness of the project for release. The implementation of this Best Practice seems to be the same for any project whether software engineering or modeling. The only differences in implementation are the points where formal defect discovery activities are planned.
6. **Treat People as the Most Important Resource** is sound advice for any project. We have not observed many of the poor people-management practices that motivated this Best Practice. However, it does no harm to remind us all that the project staff need to be selected, trained, and retained as if they were the only thing between here and project success – because they are! For simulation modelers, it is important to insure they have the requisite training and the appropriate software tools for statistical analysis of input and output data and a solid grounding in design of experiments. They should also be backed by visible and substantive support for the project from all levels of management.
7. **Configuration Management** is a process for insuring that the correct versions of the model, the input data, and the output results are matched. Some modelware has tools that instill a formal process of check-in and checkout on each component (model, input data, etc.). For typical discrete event simulation projects it is usually sufficient to maintain a log of file names with dates and content. Validation of model version is accomplished by including the model version/date in all outputs. Input data validation is accomplished by including a list of all input values in the output.
8. **Manage and Trace Requirements** is critical to success of the project. Requirements are converted into model logic, input data definitions, assumptions, and some are not included. The project manager needs to be able to demonstrate how each requirement was satisfied or not satisfied. Each requirement also needs to be mapped to one or more test cases. Tracking them is the only way to know when the project is complete. A spreadsheet works reasonably well to manage requirements.
9. **System-based Software Design** is an admonishment to make sure the programmers and systems engineering for the project are included in the design phase. These skills will provide guidance on ease of construction, operation, and maintenance that is not usually part of the simulation modeler's skill set.
12. **Design Twice and Code Once** also applies to model development. It is always tempting to go directly from a discussion with the users to the modelware and begin defining variables, resources, and logic paths. A much better process is to develop and validate a conceptual model (boxes and arrows) iteratively; and then develop variables, resource definitions, routing paths, control logic, etc. The resulting models will be easier to maintain, simpler to operate, and more likely to meet the user's needs.
14. **Formal Inspections** should be conducted on at least the requirements, the project plan, the conceptual model, the model, and the test plans. Since inspections have been proven to be the single most effective defect detection activity, there's no reason to skip this recommendation. The effectiveness of inspections can be improved by using the excellent guidelines as found in Gilb (1993).
15. **Manage Testing as a Continuous Process** is accomplished by testing each project deliverable.
16. **Compile and Smoke Test Frequently** suggests an iterative or continuous improvement approach that has been shown to be highly effective. Whenever an additional capability is added to the model, a regression test of the prior capabilities should be done first. In this way we assure that the new capability does not detract from earlier model components and functionality.

6 IMPLEMENTATION

There may be resistance among simulation modelers to implementation of these Best Practices. In particular, formal inspections may appear to be intrusions upon an individual's responsibility. Tracing requirements may seem to be an added burden with no return on investment. Tracking defects may be another intrusion upon an individual's process. The overwhelming evidence of advantage is plentiful in the software engineering literature. The references and web sites included earlier contain excellent motivational material. We have used all of these Best Practices in both software engineering and simulation modeling projects with great success and encourage widespread acceptance in the discrete event modeling community.

7 CONCLUSION

We have mapped the best ideas on process from two domains together. Our experience indicates there is a significant improvement in the probability of project success if these Best Practices are part of the discrete event simulation project plan.

ACKNOWLEDGMENT

We would like to thank the anonymous referee who made several suggestions for improvement.

REFERENCES

Arthur, James D. and Richard E. Nance. 1996. Independent Verification and Validation: A Missing Link in Simulation Methodology? In *Proceedings of the 1996 Winter Simulation Conference*, ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J.J. Swain. 230-236. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Balci, Osman. 1997.Verification, Validation, and Accreditation of Simulation Models. In *Proceedings of the 1997 Winter Simulation Conference,* ed. S. Andradóttir, K.J. Healy, D. H. Withers, and B. L. Nelson. 135-141. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Brown, Norm. 1999. High-Leverage Best Practices – What Hot Companies are Doing to Stay Ahead and How DoD Programs Can Benefit. *Crosstalk,* October, 1999.

Cook, Dave and Les Dupaix. 1998. Life Cycle Reviews from a Software Engineering Perspective, Presented at the 1998 Software Technology Conference, May 1998, Salt Lake City, Utah.

Fagan, Michael E. 1986. Advances in software inspection, *IEEE Transactions on Software Engineering,* 12(7): 744-751.

Gilb, Tom and Dorothy Graham. 1993. *Software Inspection.* Addison-Wesley, New York, New York.

Hewitt, Willard C., Jr. and Eric E. Miller. 1999. Applying Simulation in a Consulting Environment – Tips from Airport Planners. In *Proceedings of the 1999 Winter Simulation Conference,* ed. Phillip A. Farrington, Harriet Black Nembhard, David T. Sturrock, Gerald W. Evans. 67-71. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Joines, Jeffrey A. and Stephen D. Roberts. 1995. Design of Object-Oriented Simulations in C++. In *Proceedings of the 1995 Winter Simulation Conference*, ed. C. Alexopoulos, K. Kang, W.R. Lilegdon, and D. Goldsman. 82-89. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Law, Averill M. 1999. ExpertFit: Total Support for Simulation Input Modeling. In *Proceedings of the 1999 Winter Simulation Conference,* ed. Phillip A. Farrington, Harriet Black Nembhard, David T. Sturrock, Gerald W. Evans. 261-266. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Law, Averill M. and W. David Kelton. 1999. *Simulation Modeling and Analysis,* McGraw Hill, Inc., New York, New York.

Musselman, Kenneth J. 1994. Guidelines for Simulation Project Success. In *Proceedings of the 1994 Winter Simulation Conference*, ed. J. D. Tew, S. Manivannan, D.A. Sadowski, and A.F. Seila. 88-95. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Pritsker, A. Alan B. and Jean J. O'Reilly. 1999. *Simulation with Visual SLAM and AweSim,* John Wiley and Sons, New York, New York.

Robinson, Stewart and Vinod Bhatia. 1995. Secrets of Successful Simulation Projects. In *Proceedings of the 1995 Winter Simulation Conference*, ed. C. Alexopoulos, K. Kang, W.R. Lilegdon, and D. Goldsman. 61-67. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Sadowski, Deborah A. and Mark R Grabau. 1999. Tips for Successful Practice of Simulation. In *Proceedings of the 1999 Winter Simulation Conference*, ed. P.A. Farrington, H. B. Newbhard, D. T. Sturrock, and G.W. Evans. 60-66. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Software Program Managers Network (SPMN). 1999. 16 Critical Software Practices™ for Performance-based Management. <http://www.spmn.com/critical_software_practices.html>

Withers, Brian D., A. Alan B. Pritsker, and David H. Withers. 1993. A Structured Definition of the Modeling Process, In *Proceedings of the 1993 Winter Simulation Conference*, ed G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles. 1109-1117. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Zeigler, B.P. 1990. *Object-Oriented Simulation with Hierarchical, Modular Models,* Academic Press, San Diego, California.

Zeigler, Bernard P. and Sankait Vahie. 1993. DEVS Formalism and Methodology: Unity of Conception/Diversity of Application. In *Proceedings of the 1993 Winter Simulation Conference*, ed G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles. 573-579. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

AUTHOR BIOGRAPHY

DAVID WITHERS is a Senior Manager with Dell Computer Company in Round Rock, Texas. He received a BS in Engineering from the U.S. Coast Guard Academy,

and MS degrees in mathematics and computer science from Rensselaer Polytechnic Institute. He has held a variety of management and technical positions with the U.S. Coast Guard, IBM, and LEXIS-NEXIS. His research interests are in application of simulation to the solution of business problems. His publications include contributions in the *Proceedings of the Winter Simulation Conference*, the *Journal of Computational Physics*, and the *IBM Journal of Research and Development*. He is a member of the Association for Computing Machinery, The Institute for Operations Research and the Management Sciences (INFORMS), and the INFORMS College on Simulation. He was the General Chair for the 1997 Winter Simulation Conference. His e-mail address is <David_Withers @dell.com>.