Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

HOW TO BUILD BETTER MODELS: APPLYING AGILE TECHNIQUES TO SIMULATION

James T. Sawyer
David M. Brann

TranSystems
512 Via de la Valle, Suite 310
Solana Beach, CA 92075, USA

ABSTRACT

For simulation practitioners, the common steps in a simulation modeling engagement are likely familiar: problem assessment, requirements specification, model building, verification, validation, and delivery of results. And for industrial engineers, it's a well-known adage that paying careful attention to *process* can help achieve better results. In this paper, we'll apply this philosophy to the *process of model building* as well. We'll consider model building within the framework of a software development exercise, and discuss how best practices from the broader software community can be applied for process improvement. In particular, we'll focus on the "Milestones Approach" to simulation development – based on the popular "agile software" philosophy and our own experiences in real-world simulation consulting practice. We'll discuss how thinking agile can help minimize risk within the model-building process, and help create a better simulation for your customers.

1 INTRODUCTION

Several years ago, we were working on a particularly challenging project for a major U.S. airline. The airline has entered into a recurrent contract negotiation with its pilots to define the terms of the next contract, focusing on the work and compensation rules by which the airline and its pilots would be subject to work together. The airline identified the need for a simulation-based tool that accounted for the impacts of varying behavior within the daily staff and flight activities which were thought to be significant cost drivers, including human decision making by the pilots in planning their monthly schedules. The simulation results were used to quantify the impacts of the negotiated contract work rules, providing important data to support the negotiating teams.

This was a large and difficult project, equivalent to modeling minute details within hundreds of pages of the pilot-airline contractual agreement, as well as the conditions in which those clauses would be triggered. Although we had an experienced simulation team in place, we knew that this project would be a little different. The timeline was almost unreasonably aggressive. It was impossible to identify and document a full set of project "requirements" in advance of the model development. In fact, new or changing requirements would come up almost daily, as the simulation team learned more about the project, the customer's team learned more about the capabilities of simulation, and in one or two cases new requirements came from the negotiating table itself. Parallel development tasks were mandatory due to the sheer volume of model-building work that needed to be performed. The scope was constantly changing. And the results had to be not only accurate, but easily explainable and defensible due to the high-stakes nature of how the results were to be used.

This story has a happy ending: we completed the model on time, obtained useful results for the team, and had an impact on improving the overall result of the negotiations. However, many projects have not fared so well in such an environment, with the high frequency of scope change, demanding timeframe, and management of multiple developers. Our key to success was to carefully consider the *process* of model development before we even started the project, and create a plan that would minimize risk along the way. We survived – and you can too.

2 KEYS TO A SUCCESSFUL SIMULATION PROJECT

A successful simulation modeling project contains many components, ranging from the initial stages of problem assessment and requirements gathering to model verification, validation, statistical analysis, and presentation of results.

Previous articles at this conference and others (Banks & Gibson 2001) have described these steps in detail and emphasized the importance of each step in this process. For instance, if the context of the problem is not thoroughly understood, how do you know you are simulating the right part of the system in question? If there has been no formal verification that the model is functioning as designed, how do you know the analysis of results is meaningful? If the model results have not been validated against real-world data, how do you know the model is a sufficient representation of the system being simulated?

Other tutorials and papers (Standridge et al., 2007) have and will continue to cover these critical topics. However, it is easy to skim over the fact that one of the unavoidable steps is to build the simulation model! It is often assumed that we are just inherently experts at the "model building" part of this – after all, isn't that what our college courses taught us? Building any old model may be easy, but building a *good* model isn't necessarily so.

Consider the staffing approach of a typical modeling engagement. The "senior" staff member generally works with the client to determine the scope of the problem and identify what needs to be modeled. Subsequently, the problem is handed off to the "junior" staff members to construct the simulation model, under the direction and tutelage of the wise seniors. After the model is completed (and hopefully tested, verified, and validated), the senior staff member re-enters the scene to assist with the analysis and discuss the results with the customer. While it is certainly true that experience is critical to successful simulation project management and execution, note that this staffing approach implicitly suggests that building the model is the "easy" part of the project – the part that can be handled by the less-experienced junior staff.

Building a good model is not usually easy. Granted, there are many simulation software packages commercially available that advertise ease of use for the casual business user. If you can draw a flowchart, they say, you can build a simulation model. No programming required! Perhaps there are problem domains where this holds true. However, in the hundreds and hundreds of projects our firm has completed over the last twenty years, it is rarely the case that a system to be modeled is so simple and straightforward. It would be a breath of fresh air if one of our customer's problems could be naturally and sufficiently represented as the classic "bank teller" queuing problem!

Real-world systems are complex. Entities and objects and people move around. They interact with each other. They don't always behave in the exact same way. They break down. The actions or decisions of different objects may have intended or unintended effects on others. To adequately represent this complexity in a simulation model is not an easy task. Our approach to managing complexity has been to treat the simulation model construction effort as a software development exercise, and look to the broader software community for guidelines and recommendations for process improvement.

3 MODEL BUILDING AS SOFTWARE DEVELOPMENT

In its barest form, a simulation model is a compiled software program that executes on a computer and produces output results. This is easy to forget with the proliferation of graphically-oriented simulation software applications like Arena, ProModel, Witness, AnyLogic, and many others. In the past, constructing a simulation model was the exclusive domain of computer programmers; in fact, what is commonly referred to as the first object-oriented programming language, Simula, was itself a simulation language. However, user-friendly features have been designed to make it easier and faster for the business user to design, develop, and work with simulation models. You don't have to be a computer programmer in order to create a model. This evolution has been positive for our community as a whole, opening up the possibilities of simulation technology to a much broader range of users with a broader range of technical skillsets.

But under the hood, software source code is still being generated, compiled, and executed – this is the heart of a simulation. Focusing on how the model is developed and constructed – in the same way as any software is developed – can provide insights and benefits that can make projects more successful. For industrial engineers, it's a common tenet that focusing on the quality of the process can directly impact the quality of the product. In this, case we are looking to improve the quality of the simulation construction process – how the software is being built.

But why focus on software construction at all? There are many steps in software engineering, just as there are many steps in a simulation project. Steve McConnell, author of the classic and seminal book *Code Complete (2004)*, explains:

> *The ideal software project goes through careful requirements development and architectural design before construction begins. The ideal project undergoes comprehensive, statistically controlled system testing after construction. Imperfect, real-world projects, however, often skip requirements and design to jump into construction. They drop testing because they have too many errors to fix and they've run out of time. But no matter how rushed or poorly planned a project is, you can't drop construction; it's where the rubber meets the road. Improving construction is thus a way of improving any software-development effort, no matter how abbreviated.*

Designing software is itself an exercise in managing complexity (Reeves 1992). Throughout the history of software development, formal and informal techniques have been tried, tested, and endlessly debated to help the software practitioner create more effective applications. The historical benchmark of best practices is known as the "waterfall approach" to software projects.

The waterfall approach was originally described by Winston Royce in a 1970 paper (Royce 1970) on managing the development of large software systems. Its recommendations would be familiar to most simulation practitioners as it features the more-or-less traditional steps of Requirements, Design, Construction, Integration, Testing, Installation and Maintenance. This organized and methodical approach has been successful in the construction of complex, mission-critical software applications in many industries. However, the waterfall approach has been criticized over the years (Parnas & Clements 1986; Weisert 2003) for being inflexible, and other development methodologies have emerged as a response.

The recognition of the fact that requirements can and do change during the development process ultimately led to the development of a variety of approaches falling under the general description of "agile software development". Emerging in the mid-1990s, agile development realizes that change is inevitable and focuses on an iterative approach, with each iteration essentially being a small development project unto itself. The end product of each iteration may not be something for final release to the customer, but the goal is to continually have working, and tested, versions of the software through the course of the development project. Agile processes and practices focus on the ability of a development team to respond to change, and the ability of software to accommodate change yet still remain working software (Knoernschild 2006).

In 2001, the principles of agile software development were summarized by a set of champions within the software engineering community in what is known as the *Agile Manifesto* (Beck et al. 2001). Some of the key principles can be summarized as:

- Achieve customer satisfaction through rapid, continuous delivery of useful software
- Working software is delivered frequently (weeks rather than months)
- Working software is the principal measure of progress
- Even late changes in requirements are welcomed
- Close, frequent cooperation between customers and developers

One key distinction that must be made regarding agile development (and related methodologies, such as "extreme programming") is that it is absolutely not the same thing as having no process at all, or "cowboy coding". While in both cases, there may be a relative lack of documentation, there are still important processes to be followed within agile development, as we describe in the following sections.

4 AGILE SIMULATION: THE "MILESTONES APPROACH"

What makes the agile software development approach appropriate for a simulation development project? Why not use the waterfall approach, or a modified waterfall to allow for some iteration? Because in the real world, the requirements for a simulation can and do change during the course of the project, and agile approaches are designed to help manage that change. The customer's priorities may change. You may learn something during the simulation development that changes certain assumptions. Something you may have thought was absolutely critical before the project began may turn out to have minimal meaningful impact on the results. No matter how thorough the requirements gathering phase of a project, it is very rare that a functional specification can be created that is completely set in stone.

This doesn't mean that a requirements specification is a bad idea! It helps both the simulation practitioner and the customer understand and limit the scope of a project – which is important, assuming there are limits to the available time and budget to spend on a problem. It helps to think of a specification as a starting point for this continually evolving development process.

The agile philosophy suggests that changes to project requirements should be expected, and accounted for as a natural part of the software project process. In our consulting practice, this philosophy has been embraced in what we call the "Milestones Approach" to simulation development. The Milestones Approach is intended to divide a large and complicated project into smaller phases with clear objectives that can be a) successfully navigated and managed by the development team, and b) quickly demonstrated to communicate progress to the customer. The objective is to improve the quality of the end product by separating the project into more manageable pieces, and provide an opportunity to evaluate the development process at every step.

Each milestone is a self-contained, work-in-progress version of the overall project. Each milestone adds detail to what has been accomplished in previous phases. Each milestone serves as a review point where the team can step back and reexamine the development schedule, methods, testing, and usability of the model. Each milestone results in a working deliverable so that progress can be demonstrated and communicated to the customer. Documentation of each milestone can be delivered to the customer to jointly track project progress. Inevitably, each milestone review with the customer leads to new or modified requirements.

For example, if processes A, B, C, and D all need to be modeled for a system representation, it might be tempting to start by implementing all of the minute intricacies within process A. Instead, using the Milestones Approach, a basic model is constructed that includes some high-level but complete representation of A, B, C, and D. This Milestone 1 version of the model contains a relevant subset of input parameters and produces output results. By evaluating these results, further guidance can be gained on which of the four processes is more important to investigate in more detail.

This doesn't mean the details of process A or any other process should be ignored. The details are acknowledged and written down, and a conscious decision is made about which project milestone is the right one for scheduling the implementation. In our experience, we've seen many failed projects where too much attention was paid to minute details too early in the project timeline.

5 WORKING WITHIN MILESTONES

The typical lesson plans used to introduce a first simulation package, whether in a college course, research, or simulation practice, also reflect elements of agile development. For example:

- Lesson 1: Basic Source, Queue, Delay, Sink model (or maybe flowing from A to B on a conveyor).
- Lesson 2: Add a Resource that needs to be claimed before starting the Delay.
- Lesson 3: Add branching logic so that there are two Queue-Delay pairs and the entities chose between them based on some condition (and so on)

Simulation training takes that approach to adding complexity to the model because the student is incrementally acquiring the needed skills and knowledge to model that complexity in a given tool. That approach follows many of the same rules as agile development: for each lesson (milestone) there is a fixed set of functionality to be added, and there is a working, demonstrable product at the end of the milestone. Those same principles should guide the development of the milestones for a much more complex simulation model.

Important concepts within the Milestones Approach include:

- Plan the work
- Frequent iterations
- Frequent testing
- Frequent review

Think of all of the dozens or hundreds of individual tasks and features that need to be executed in order to say that the model is complete. The first step is to come up with a plan of attack. The Milestones Approach encourages setting the details aside and starting with a broad view of the architecture. What are the primary functional or logical components within the system to be represented? List them. Also list any other elements that may be needed to complete the simulation project: inputs, outputs, user interface, formal testing plans, creating an installation package to deliver to the user, and any other relevant work tasks.

Then, divide the project timeline into a series of individual milestones. How many and how frequently depends on your overall project schedule and scope. We've found that 1- to 2-week milestones generally work fairly well with our project teams – it's long enough for significant development and testing to occur, but short enough that we're not waiting too long to establish a review point and potentially make a course correction.

At this point, a planning matrix can be created, as illustrated in **Error! Reference source not found.** below. Project components represent the rows, and milestones represent the columns. Starting with Milestone 1, fill in the individual detailed tasks and features step by step. Prioritize the implementation of those details according to their importance to the overall goals of the model, and organize the project into sequential milestones over time.

Each milestone has an objective. In the first milestone, you are trying to create a basic framework for the rest of the model development – the "skeleton" of the project if you will. In subsequent milestones, the objective is to add in a little more detail to each of the functional components. One way to think about this is to examine each piece of logic and ask: does this process occur all of the time, some of the time, or rarely? Is it a "special case" that happens only a small percentage of the time? Rare events doesn't belong in an early project milestone.

For instance, it is common in our early project milestones to set processing delay times to be constant values. This enables easier testing to verify that the correct system delay times are being executed in aggregate. In later milestones, we can add in variability to processing delays. Another example is modeling system failures and breakdowns – in early milestones, we make the assumption that the processes will work perfectly with no breakdowns. Once we have tested and verified that the process flow of the model works correctly in the base case, we can add in probabilistic failures.

5.1 What to Consider When Setting Milestones

A milestone is intended as a stopping point and a review point. At the end of each milestone, you should have produced a working model or program that you would feel comfortable demonstrating to the customer. At this point, it is beneficial to step back and put yourself in the customer's shoes. Can you see satisfactory indication of progress at this point? What does the tool look like? How easy is it to use? Do the output results make sense, given the fact that some details may be missing? When caught up in the hard work and intricate details of programming models, it is sometimes easy to forget about the big picture and the customer's perspective.

Conducting a series of iterative milestones also helps the soft side of simulation practice – the psychology of the development team. Completing a milestone reinforces the attitude that the team is making consistent and real progress. In big projects, it's easy for a development team to feel overwhelmed by the sheer quantity of work remaining, and lose sight of the light at the end of the tunnel. Periodic endpoints communicate a sense of accomplishment, both to the customer and to the internal team.

Within each milestone, it's important to limit the scope. Sometimes it may seem tempting to add in new features to a milestone while it's in progress, even if it wasn't originally planned for that milestone, particularly if it's easy or quick to do. But every new feature implemented is a new feature that needs to be tested – and a new potential risk to break some other piece of previously working code.

Try to focus on coding and thoroughly testing the limited feature set that was planned within that milestone. A milestone is small and self-contained for a reason: it reduces the number of variables that could affect the end result, and spreads out the required testing throughout the project timeline. By putting too many features in at once, the overall effort required to get it right only increases.

5.2 Quality Control Within the Milestones Approach

Two key elements of the model building process can significantly enhance the quality of the simulation product within the Milestones Approach: code walkthroughs and system testing at the end of each milestone. While these will be discussed more thoroughly in future papers, they are worth outlining briefly here.

Code walkthroughs are group meetings where one developer will walk the rest of the team through key portions of the code they have been working on, with the overriding goal being to improve the quality of the code under review. Ideally, walkthroughs take place at least once per milestone, perhaps more frequently depending on how rapidly coding is occurring. Team members take responsibility to review the code in advance of the meeting so that they can be prepared with helpful ideas. But decorum is advised: Differences in coding styles can lead to arguments, and so the discussions should be focused on logical issues or easily overlooked errors, and a designated moderator can play a valuable role. The objective is to work together to create a better product, not to evaluate the individual.

Frequent testing is also a critical aspect of the Milestones Approach. Effective testing procedures for simulation models could be a subject for an entire paper, but to sum it up simply: Test early and test often. Iterative milestones provide a natural point in the project's timeline for testing the features developed during that milestone.

As the features in Milestone 1 are completed, the tests for Milestone 1's features are created and executed. As the features in Milestone 2 are completed, not only are Milestone 2's new features tested, but Milestone 1's tests are re-run to ensure that none of the changes have adversely affected something that was working before. This practice is called *regression testing* – the selective retesting of a system or component to verify that modifications have not caused unintended effects, and that it still complies with the specified requirements (Geraci 1991).

For a complete discussion of *how* to perform verification and validation of simulation models, please refer to Robert Sargent's classic papers on the topic (Sargent 2007). This section has suggested *when* to perform that verification and validation during the course of a project.

6 PROJECT EXAMPLE

Let's consider how the Milestones Approach might be used to plan a simulation project for a partially automated order fulfillment facility. We might start by outlining the major functional components within this facility, such as Picking, Packing, and Shipping. We also know that there must be some demand for orders to be fulfilled, and that there is some movement or flow of objects between each of these components. These become the major elements of our planning matrix, shown in Table **1**.

Table 1: Sample Milestone Planning Matrix

	Milestone 1 **Objective:** Basic system running **Target date:** +2 weeks	**Milestone 2** **Objective:** Conveyors and UI complete, adding detail to Picking and Shipping **Target date:** +4 weeks	**Milestone 3** **Objective:** Model complete and ready for analysis **Target date:** +6 weeks
Order generation	Simple orders with 1 line per order	Allow multiple lines per order	Create orders based on probability table
Picking	Continuous Picking of orders	Break order Picking down into waves	Add details of Picking resource
Packing	Black box delay	Black box delay	Add details of Packing resources
Shipping	All orders routed to same dock	Add details of shipping resources, all applied at same dock	Orders routed to variety of shipping docks based on destination
Material Flow	Basic conveyor system constructed and functional	Merge/split logic added to conveyors	<no additional functionality>
Model Testing	Basic "what goes in must come out" testing	Structured testing for new outputs, User Interface, and Picking, Shipping logic	Model passes all structured unit and system tests
Inputs	Read basic parameters for order generation	Read parameters for wave Picking, Shipping resources, and all conveyor-related inputs	Add Picking and Packing resource read routines, and parameters for Order Generation
Outputs	High-level statistics: # Orders picked, packed, and shipped	More detailed statistics: resource utilization, orders per wave	All required outputs implemented
User Interface	Majority of user interface mocked-up	User interface functionally complete	<no additional functionality>

Once the overall planning matrix has been developed, the evolution of each individual component over the course of the project can be forecasted. For example, focusing on the Order Generation component, the Milestone 1 version of the model could initially be constructed to read in a simple set of orders, where each order contains only a single line. In Milestone 2, the logic could be extended to allow multiple lines per order. And in the final Milestone 3, the order generator component could be completed by adding an algorithm that can create synthetic orders based on user-specified distribution profiles.

Each functional component is then approached in the same fashion, carefully analyzing how additional detail could be added to the component's model representation sequentially, in order to arrive at the desired end state. As each component is evaluated, its interaction with all other components needs to be considered, so that the end of each milestone represents a mini-product that is functionally complete within the bounds of its own objectives.
A few items to highlight in this hypothetical plan:

- The complete user interface is prototyped during the first milestone. This facilitates clear communication with the customer about the system "switches and dials" that will be available for the analysis – a common source of change requests in a project's lifecycle.
- There are separate components listed for creating the model inputs and outputs that are relevant to each milestone, not just the simulation logic that needs to be coded. This reinforces the recommendation that each milestone should be treated as a self-contained project in and of itself.
- The level of detail in the simulation inputs mirrors the development plan of the functional components.
- The level of detail in the output statistics increases along with the level of detail in the model logic.
- It is recommended to defer lower priority tasks to future milestones – sometimes they may not need to be modeled at all. For example, the Pack-

ing component of this hypothetical model was not an item of large concern, and so it was left as a "black box" delay for the first two milestones.

- Some components, as illustrated in the Material Flow and User Interface components, may be 100% complete prior to the final milestone. This might be the case with model components that are critical to the working of the rest of the model or that it may be important to have extra testing time for (such as the user interface).

7 CONCLUSION

Building a good simulation model is not an easy task. By focusing on the quality of the model construction process itself, you can achieve a higher degree of quality in the end result – not just the simulation that is created, but the answers and insights that you learn from using the model to perform analysis. Building simulation models can be treated as a software development exercise, and best practices from the software community can be applied to help improve the model-building process.

We've found that the agile philosophy, an increasingly popular movement within the software development community, offers many best practices that translate well to a typical simulation project. Our Milestones Approach takes the key agile concept of frequent, iterative deliverables and frames it in the context of executing modeling projects. The advantages of the Milestones Approach were discussed, and a project example was provided that indicates how this might be put into practice in modeling a hypothetical order fulfillment center.

By applying the Milestones Approach consistently within our consulting engagements, we've found that focusing on an improved process results in better quality project deliverables, improved team performance and morale, and importantly, more satisfied customers.

REFERENCES

Banks, J. ,and R. R. Gibson. 2001. Simulating in the Real World. *IIE Solutions,* 33(4) (April): 38-40.

Beck, K., Beedle, M. and many others. 2001. The Agile Manifesto. Available via <http://agilemanifesto.org/principles.html>

Geraci, A. 1991. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries.* New York: The Institute of Electrical and Electronics Engineers, Inc.

Knoernschild, K. 2006. An Agile Resolution. *Agile Journal,* December.

McConnell, S. 2004. *Code Complete: A Practical Handbook of Software Construction*, Microsoft Press.

Parnas, D. L., and P. C. Clements. 1986. A Rational Design Process: How and Why to Fake It. In *IEEE Transactions on Software Engineering*, 12: 251-257. IEEE Press.

Reeves, J. W.. 1992. What is Software Design? In *C++ Journal.*

Royce, W. 1970. Managing the Development of Large Software Systems. In *Proceedings of IEEE WESCON* 26 *(August):* 1-9.

Sargent, R. 2007. Verification and Validation of Simulation Models. In *Proceedings of the 2007 Winter Simulation Conference*, ed. S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 124-137. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Standridge, C. R., D. A. Finke, C. Jurishica, D. M. Ferrin, and C. M. Harmonosky. 2007. What I Wish They Would Have Taught Me (Or That I Would Have Better Remembered!) In School. In *Proceedings of the 2007 Winter Simulation Conference*, ed. S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 2315-2321. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Weisert, C. 2003 Available via <http://www.idinews.com/waterfall.htm> Chicago : Information Disciplines, Inc. [accessed March 2, 2008]

AUTHOR BIOGRAPHIES

JAMES T. SAWYER is an Assistant Vice President and Senior Professional partner at TranSystems. He is the former Chief Technology Officer of Automation Associates, Inc., a leader in simulation for the global supply chain, which was acquired by TranSystems in 2005. His technical specialty is the design and development of simulation-based software solutions for the transportation and logistics industries. He is the software architect of the *Modeling Studio*, TranSystems' standardized simulation application and user interface framework enabling rapid development and deployment of simulation projects across teams and industries. His current role is to lead development teams in simulation modeling and software projects, and to foster best practices in technical project execution. Mr. Sawyer received a B.A.S. in Mathematical & Computational Sciences at Stanford University, and an M.S. in Industrial & Systems Engineering at the Georgia Institute of Technology. <jtsawyer@transystems.com>

DAVID M. BRANN joined TranSystems | Automation Associates in 1999 and has over ten years experience in designing, developing, and managing the implementation of simulation-based software solutions across a wide variety of application domains. He has experience using a variety of simulation languages as well as supporting software such as Visual Basic, Java, C/C++, and Microsoft Applications (Access, Excel, Visio, etc.) Mr. Brann has

prior academic experience that includes software development in several languages for the purposes of research in human-centered intelligent systems and object-oriented simulation. In addition to his work in managing and developing custom software solutions, Mr. Brann is also responsible for providing training courses on a variety of software tools, including TranSystems' own custom packages and AnyLogic simulation software. <dmbrann@transystems.com>