

Chapter 2

A Collaborative Process for Multidisciplinary Group Modeling Projects

James D. Westervelt and Bruce Hannon

2.1 Introduction

Although a virtually unlimited number of ecosystem management issues may be illuminated using small, expedient simulation models developed by one person or a few, there are many cases where much more complex problems can be efficiently modeled by a relatively larger team that spans several disciplines or technical domains. This chapter describes a process for conceiving, coordinating, and launching such a model development initiative using the same simple software platforms described elsewhere in this book.

We have successfully taught ecosystem modeling to groups of university students as a three-stage process, with a sequence of steps comprising each stage. By adhering to this process, our students have developed highly utilitarian ecological simulation models that are based on real-world data and specialized technical expertise. Many of the models documented in this book were developed by students. Our model-development process can be outlined as follows:

1. Prepare to model
 - (a) Identify objectives and scope
 - (b) Identify available resources (personnel, expertise, time, software, hardware)
 - (c) Consider group dynamics (including ownership issues)

J.D. Westervelt (✉)

Construction Engineering Research Laboratory, US Army Engineer Research
and Development Center, Champaign, IL, USA
e-mail: james.d.westervelt@usace.army.mil

B. Hannon

Department of Geography, University of Illinois,
220 Davenport Hall, 607 S Mathews, M/C 150, Urbana, IL 61801, USA
e-mail: bhannon@illinois.edu

2. Construct the model

- (a) Set modeling constraints
- (b) Conceptualize the full model
- (c) Design submodels
- (d) Construct a full dummy model
- (e) Construct the submodels

3. Integrate the model

- (a) Debug NetLogo¹ compilation errors
- (b) Debug errors in model logic
- (c) Demonstrate to end users

This process applies irrespective of the modeling software environment that is used. Each of these steps is discussed below.

2.2 Prepare to Model

2.2.1 *Identify Objectives and Scope*

The first step for successful multidisciplinary dynamic modeling efforts is to review the objectives and scope of the project. This involves answering the following essential questions:

- *Who is the end user?* This question is often glossed over because model developers often assume that the end user is exactly like them. A model's utility can be significantly limited when the needs and applications of a prospective external community of users are not fully considered at the outset. This often becomes clear to the model developer only when the model is demonstrated to others.
- *What does the end user require of the final model?* Determining what the identified end user expects from the modeling effort requires substantive communication among the modeling team and the end user. The answers to this question are not always immediately apparent to the prospective end user, and they often need to be drawn out through direct and open communication between modelers and prospective end users.
- *How accurate do the output requirements need to be?* There is no single correct answer to this question. For some end users, it may be sufficient for the model to be capable of generating relative or suggestive output to show trend directions. Other end users may require highly accurate data for purposes of fine-tuning land

¹This text assumes that the modeling group is using NetLogo (Wilensky 1999) as its collaborative development platform.

management scenarios. Again, the most suitable answer is arrived at through direct, substantive communication between the modeling team and prospective end users.

2.2.2 Identify Available Resources

This task also can be aided by answering certain essential questions related to personnel capabilities and availability, data requirements, and computer technology, as discussed below.

2.2.2.1 Personnel Capabilities

- *What expertise is available through the modeling group?* If the group has the right mix of collective experience, model development can proceed rapidly and with great efficiency.
- *What actions are needed to fill any knowledge gaps in the team?* Depending on the gaps identified, the group may either need to recruit outside expertise or identify supplemental training needs for various team members.

2.2.2.2 Participant Availability

- *How much time will each participant be able to provide the modeling effort?* Regardless of how fast and proficient each team member is, and how well organized the team is, a participant's lack of availability during critical "windows of opportunity" for collaboration can delay progress. Availability of team members is easy to overlook because it may seem imponderable at the planning stage.
- *What time frames are available for inter-team coordination?* For interdisciplinary efforts, the viability of a group member must be assessed with respect to how much time the member can be available to coordinate with the others.

2.2.2.3 Available Data

- *Have other models already been developed for the subject environment?* Find out what other models are already available and how they may apply to the current problem. Investigate what existing models can teach the team about modeling the environment and about problems or limitations yet to be addressed.
- *What data are available?* Determine whether the model can simulate the subject environment to the required level of accuracy using readily available data.

If insufficient data are available to produce the level of accuracy needed, the responsible person needs to determine whether it is feasible to obtain more funding or work time to fill the data gap through additional fieldwork or research.

- *What is the full set of data requirements for the model?* Thoroughly consider all forms of data needed, not just technical information about the biome. A spatially explicit model requires spatial data such as raster maps, satellite imagery, vector maps, polygon maps, and point data (representing the location of data sampling points).

2.2.2.4 Computer Technology

- *What computer hardware is available?* Review the inventory of hardware available to support the project locally. Consider whether local network or Internet access is needed, and whether the available connections have sufficient bandwidth to support communication between connected machines.
- *What software capabilities are available?* In addition to selecting the software modeling environment, consider what ancillary software may be needed to support the modeling effort.
- *Do the benefits of the identified computer technology justify its cost?* It may be tempting to acquire sophisticated information technology that requires heroic effort to apply. This temptation may arise from the desire to do the best possible job, or simply to extend the joy of modeling as long as possible. From a business standpoint, this approach may not provide a sufficient benefit, so a less expensive approach (in terms of money, time, or both) may be optimal.

2.2.3 Consider Group Dynamics

Interdisciplinary efforts have special management needs. Traditional management hierarchies must be deemphasized in favor of more unmediated coordination and cooperation among team members. Individuals from different disciplinary backgrounds often base problem-solving on different paradigms, which makes some difficulty in communication likely. Such difficulties, however, provide an opportunity for significant “cross-pollination” of knowledge within the team. Focusing specifically on group management considerations that pertain to model building, several issues that have a direct bearing on integrating individual efforts into the group product must be recognized and addressed.

2.2.3.1 Model Development and Integration Responsibilities

Large, complex, multidisciplinary modeling efforts undertaken by a team must be split logically into model subcomponent development tasks. The role of each task in

the project is defined in terms of the scope of the entire project. Each model subcomponent is completed through individual effort, imagination, and expertise. Development of the subcomponents begins with the definition of subcomponent requirements, continues with coding that is often accomplished in isolation, and completed with refinement efforts during full model integration. It is crucial that the owner of each subcomponent be involved during all phases of integration. An individual's temptation to "turn over" a model component should be discouraged until the entire project is completed. If a component that has been "completed" prematurely and ultimately needs refinement or revision based on overall model development, it is almost always more efficient for the author of that component to make the changes. In cases where it is necessary to assign a different team member to make the revisions, it is often better to completely rewrite the subcomponent than to spend much time trying to understand the original. For the sake of time, cost, pride in accomplishment, and avoidance of disruption, individual responsibility for subcomponents and submodels must be maintained throughout entire project.

2.2.3.2 Scope of Subcomponent Development Efforts

Each individual developer must understand that his or her task is the development of a submodel that provides essential outputs for use by the full model, not a fully functional model in its own right. Developers also should understand that they need to meet, but not exceed, the functional requirements of the subcomponent. A subcomponent that does significantly more work than is needed by the full model can drain developer resources and also produce a model that runs slower than what is acceptable.

2.2.3.3 Scheduling

People work best within the context of achievable expectations. Schedules define expectations from the perspective of available time, energy, and ability. A team must work off a common schedule that sets achievable goals within realistic timeframes. A feasible yet challenging schedule must be developed with and clearly communicated with the team. "Plan the model and model according to the plan."

2.2.3.4 Leadership Among Equals

A nonhierarchical, multidisciplinary team can be difficult to coordinate. As a practical matter, someone on the team should be responsible for pulling the team together as a working unit based on his or her understanding of everyone's individual personalities, expertise, and motivations. When a difference of opinion cannot be resolved by group consensus, someone must provide leadership in order to move forward. The designation of an effective team leader may be the most important decision

made during the entire modeling project. Ideally, an effective team leader will be able to work well with all members of the team on both a personal and technical level. The person must fully understand the modeling team's overall objective, be well versed in the modeling process, and be trusted by the group to exercise impartial, well-informed judgment to avoid or resolve conflicts.

2.3 Construct the Model

2.3.1 *Set Modeling Constraints*

2.3.1.1 Potential Model Components

A large array of model components is available to the ecological modeler. Decisions made in the choice of hardware and software, however, will limit the range of options available. Typical model components to consider are briefly described below.

- *Landscape patches.* Landscapes are now recognized to be important variables in models that simulate the movement of individual organisms, structural changes in ecosystem boundaries, or the movement of air and water. Dividing the landscape into grid cells, hexagons, or irregular polygons is an effective way to capture some spatial information. For the sake of simplicity, most models are designed to use, at a fixed resolution, data in only one of those landscape patch formats. Landscape resolution is often chosen with respect to the operational time step incorporated for simulations.
- *Linear objects.* Some spatial structures are most efficiently simulated as linear objects. Examples include streams, rivers, and most of the built environment such as roads, fences, buildings, and parking lots. Incorporating both linear objects and landscape data stored as rasters or hexagons is complex, but it is often unavoidable. Just as different data storage techniques are appropriate to different data formats, so are different modeling techniques appropriate depending on the model output requirements.
- *Discrete mobile objects.* If individual people, groups of individuals, vehicles, or individuals of an endangered species are key variables in the model, they must be represented as discrete mobile entities. Such objects must be able to disconnect from a location in the landscape space and "reconnect" in an adjacent space to interact with the environment there.

2.3.1.2 Potential Model Interactions

There are a very large number of potential interactions among model components, so the following list indicates only broad categories of interaction. The modeling team must identify the model interactions that will best simulate the system.

- *Raster geographic information system (GIS) interactions.* Classes of traditional GIS interactions have been developed and discussed by Tomlin (1990). These include the following:
 - Simple location-by-location overlays that can be expressed with mathematical equations, using maps as variables. These can be used to find locations that meet certain local criteria; find correlations or potential local impacts related to a proposed change to the landscape; or transform a set of maps (e.g., slope, aspect, soil characteristics, rainfall) into new interpretations using mathematical relationships (e.g., soil erosion potential based on the Universal Soil Loss Equation).
 - Near-neighborhood operations at computer output as a function of the state of small regions surrounding each map location. These can be used to compute slope or aspect as a function of the elevations surrounding each map location, determine direction of a flow such as water or air, or identify areas where information changes rapidly (e.g., edge detection).
- *Cellular automata interactions.* Individual cellular automata (CA) comprise a two-dimensional surface that changes over time. The changes are driven by equations that generate the future state of each cell (i.e., location) as a function of the current state of that cell and its nearest neighbors (either four or eight depending on system design). For exploring the qualities and characteristics of the cellular automata approach, cells are often assigned to a limited number (fewer than 256) of states. Relaxing this limit to accommodate a large number of variables results in the type of models described in this book: models with fixed time steps that run simultaneously for a number of land parcels arranged in regular grid cell arrays. Each cell is treated as a homogenous system that can be influenced in each time step by its own state and the state of all its adjacent neighbors.
- *Vector GIS interactions.* Landscape information stored as polygons and linear features can also represent objects that interact and move such as traffic flow patterns along roadways or the hydrologic activity of stream or river networks during unusual storm events. Entities like cities, parking areas, private land, or stable ecosystem regions are most efficiently stored as polygon data elements, and they can be conceptually easy to model as distinct entities.
- *Mobile object interactions.* Some models require distinct entities that move across the landscape. Examples include individual members of an endangered species, vehicles moving about a landscape, or a group of individuals that moves collectively in close geographical contact.

These broad classes of interaction can combine into compound or hybrid types of interaction. Animals modeled as mobile objects must interact with water found in streams that are modeled as vector entities, and also with vegetation that is modeled as a component of a cellular model fixed in space. Entities can communicate with each other through sounds, propagules, pheromones, and waste gases that disseminate through the modeled space. It can be seen that the range of potential interactions is quite large. Generally, the group's options for modeling will be greatly constrained by the selected modeling environment and available computer hardware.

2.3.1.3 Simulation Timeframe

Once the purpose of the model has been determined, it should be relatively straightforward to identify the optimal timeframe for simulations. The timeframe is determined on the basis of informed guesses about how rapidly the predictive power of the model will decay over simulated time. This same issue impacts the accuracy of the sophisticated large-scale models used to forecast weather patterns; even state-of-the-art models lose predictive power the longer the simulation runs. There may be exceptions to this, however. For example, spatial models may show stability at a gross scale while showing apparently random output at a detailed scale. That is, the overall pattern may remain the same over time, but details about the location of the pattern may change with different simulation runs. In general, the timeframe is determined directly by end user requirements.

2.3.1.4 Time Step Options

The simulation proceeds from a given starting point to the end of the timeframe identified above. Simulation time may proceed according to fixed time steps, variable time steps, or the occurrence of specified events.

- *Fixed.* This is conceptually the simplest approach, but functionally it is the most limiting. The model runs with a set time step, such as 0.25 or 1, which can represent days, months, years, or other measures of time. A known time step simplifies the model because all equations are generated with respect to the same time step. A fixed time step, by definition, cannot accommodate variability in the system. If the time step is weekly, for example, then the model cannot capture daily changes in temperature, moisture, or plant growth. Similarly, a daily time step would miss the effects of a flash flood that may take only minutes to cause great devastation to vegetation.
- *Variable.* There are two options available for using a variable time step. One option is to set the time step to be long, initially, but also allow it to be modified dynamically as changes occur and are detected within the model. In the case of the flash flood example given above, when the storm occurs the model would detect rapidly changing activities, stop the simulation, back it up, and resume the simulation using an appropriately smaller time step. The second option is to assign different fixed time steps to different parts of the model. This approach requires less computational power while maintaining the relative simplicity of fixed time steps.
- *Event driven.* This approach advances time not by steps but according to a calendar that schedules specific events. A plant submodel may execute plant growth and then schedule itself to be updated at some later time based on its own rate of activity. A storm submodel would be programmed to run at a specific time, and while it runs it can interact with the plant model and schedule the plant submodel to accelerate growth in response to the influx of water. This approach is most

attractive for models designed for limited computing resources. From a modeler's perspective, however, it is the most time-consuming approach because it requires significantly larger simulation models than are needed using the other two approaches.

As can be seen, one of the main criteria for selecting an approach is the capabilities of available hardware and software. When this criterion is not critical, the selection will be made on the basis of the costs and benefits of the alternatives.

2.3.1.5 Spatial Resolution Options

The resolution of the landscape's spatial representation must be considered in terms of the modeling objective, model subcomponents, and technology constraints. Two major aspects of dynamic, spatially explicit ecological models are the spatial distribution of model components and the effects of the space on model component interaction. The resolution of space in this type of model is as important as the resolution of time steps. Most modeling environments divide space into a checkerboard-type grid surface using cells of uniform size. If the cell size is too large, the implications of the spatial arrangement of objects in the system can be lost. As cell size is reduced, more computational power is required to support the model. Another consideration is how the cell size relates to agent activities and time steps; without a logical correlation among the three, the task of developing the model logic becomes much more difficult. In the case of an animal that covers up to 1 ha in a time step, for example, a cell size smaller than 1 ha will require the modeler to track the several cells comprising the specified 1-ha area and to ensure that the animal agent interacts with all of those cells during the time step. The development of the modeling logic to address that situation could be avoided simply by assigning the cells a size of 1 ha. That said, there is no universal formula for assigning cell size because other factors may equally influence the landscape surface scaling decisions. Three general terrain resolution schemes may be applied to simulation model design:

- *Fixed.* The terrain features assume a fixed resolution that can be constant across the landscape representation. A regular array of square grid cells or hexagons is commonly used in spatial simulation environments. These have the advantage of being conceptually simple as models need not account for different or changing resolutions.
- *Hierarchical.* Models that simulate activities occurring at different spatial resolutions may adopt a spatial data structure that maintains information in a hierarchical manner. Each cell or hexagon can be iteratively decomposed into increasingly smaller components. Large entities (e.g., weather systems, flocks of birds, clouds of spores or pollens) can move rapidly across the system using relatively long time steps and large spatial patches. Smaller entities (individuals or vehicles) can operate at smaller time steps and smaller patches. In this scenario data are maintained simultaneously at varying scales.

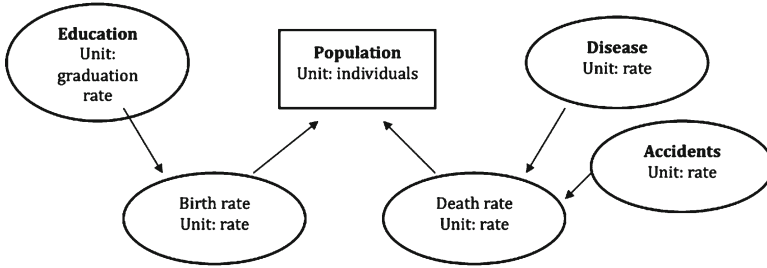


Fig. 2.1 Early state of model concept

- **Variable.** In the case of large objects that move slowly across the landscape (ecosystems, roaming herds of ungulates, or populations of invading species), there may be advantages to maintaining the entity as a single whole while retaining detailed spatial structure that defines the dynamically varying extent of the entity. This type of operation requires maintenance of the spatial extent at a fine resolution while allowing simulation of the object dynamics at a grosser resolution.

As is true with other considerations discussed above, the modeling staff will generally find their options constrained by the software and hardware limitations.

2.3.2 *Conceptualize the Full Model*

The full model concept begins with a focus on the model output requirements. These will almost always take the form of time series output showing the dynamic status of something in the model. The output series might track the status of an endangered species, property values, soil depth, land use patterns, ecosystem health, or the position and paths of vehicles or individual organisms. In all cases, a firm set of output requirements must be specified, and they will drive all subsequent decisions. First the outputs are documented as diagrams on paper, whiteboard, or computer screen (see Fig. 2.1). These represent the model's first *state variables*, parameters whose values will change over time. Each variable must be associated with a unit of measure. Some units will be straightforward (e.g., dollars, weight, mass, count), but others may be challenging. For each of these output variables, the group identifies the factors that directly influence the state of the variable. The diagram is revised as these factors are captured, with arrows showing the state variables that are affected.

At this point the group may have diagrams of system components with arrows as in Fig. 2.1. Boxed components are state variables; circled variables with arrows coming to them are calculated at each time step; and circled items with no arrows coming to them are model parameters (i.e., fixed variables). Equations will be written for every component in the diagram that has incoming arrows, so it is

important to begin ensuring that the units associated with the connected components are compatible.

This conceptual model grows over time. Participants may become anxious about whether there is enough time, funding, energy, or information to complete the model. Once a model is captured as a computer application, sensitivity experiments can identify which model values, components, equations, and logic affect model outputs to greater and lesser extents. It has often been repeated that a model is done when nothing else can be taken out of it. Although the model remains conceptual at this point, it is important that developers continually perform conceptual sensitivity analyses to discover which model components are clearly not likely to significantly affect model outputs. With judicious pruning and reference to the core project objectives, the conceptual model can be kept within limits that allow its completion using the available time, energy, and expertise. The process proceeds by looking at the key input variables that might be used to compute the state of the primary outputs desired; team members look in turn at each of these variables and determine key inputs required to compute their changing states. The group accomplishes this task iteratively until the participants are comfortable that the conceptual model is sufficiently complex to answer the primary questions of importance to the end user, but is still achievable within the identified constraints.

To summarize, the following steps are required to fully conceptualize the model:

1. Identify and discuss the primary output desired. Begin with the questions that will be asked of the completed system.
2. Discuss the model inputs that might be required for generating accurate state changes in the primary state variables.
3. Perform a conceptual sensitivity analysis for each potential input to help prioritize them.
4. Repeat the previous two steps iteratively for each important input until the model is agreed to address all primary end user questions and to be feasible to complete given the availability of project resources.

2.3.3 Design Submodels

If the model is sufficiently large, development efficiency will require that it be divided into submodels. Consider the design of a new automobile engine. The process may involve the expertise of dozens of people and it would be inefficient for all of those people to collaborate in one room on the design of every component. Instead, the design group is divided into smaller units with specific expertise to independently design engine subsystems, such as fuel delivery, ignition, cooling, exhaust elimination, and noise management. Each subsystem team is given specific parameters to ensure that the subsystem performs as specified, fits the available physical space, and is properly supported by other subsystems. The steps explained below will help guide the effective development of submodels.

2.3.3.1 Identify the Submodels

Once conceptualization of the full model is complete, responsibility for the development of the submodels is assigned. Submodel teams then divide their responsibilities among individuals, who clearly understand their tasks in the context of group needs and the overall requirements of the final model. Each team retains primary responsibility for its submodels throughout the life of the project. Thus, this decomposition of the full model must be accomplished with close attention to team member:

- Expertise
- Availability
- Learning requirements

This stage often proceeds smoothly since the capabilities of each participant are often expressed during model conceptualization.

2.3.3.2 Set Submodel Requirements

This portion of the work starts with further conceptualization of the submodels, with the immediate objective being to identify submodel input requirement that will need to be satisfied by other submodels. This process becomes an iterative conversation among all submodel groups. Through identification of submodel input requirements and submodel output possibilities, the teams work out design and development contracts with each other. Stated concisely, the three steps are for teams to:

1. Identify external input requirements for their submodels.
2. Identify potential outputs of their submodels.
3. Agree on contracts with all other submodel teams based on the required inputs and outputs.

The contract must identify publicly available state variables, variable units, and variable resolution requirements. Debate among teams will likely proceed through several iterations before a consensus is reached. It is important that this debate and the resulting contract be taken as a firm obligation. If a team labors to develop a submodel based on the agreement that a key input will be available, then that input must be made available. Failure to meet these obligations can severely weaken team cohesiveness and jeopardize successful completion of the model.

Table 2.1 illustrates a representative, partially completed “submodel team agreement table.” The column headings are filled with parameter name, associated unit, variable type, and—grouped at the right—the submodel team names. The row headings do not list all state variables in the model, but only those that are shared across models, which are those that are initialized and maintained by the submodels. The agreements captured in this table are (1) the variable names, (2) designation of the submodel responsible for each variable, and (3) the units to be applied. In some cases it is also important to include an agreement on the update frequency and the

Table 2.1 Submodel team agreement table

	Units	Type: S-state, V-variable	Weather	Reservoir	Agriculture	Population	Economy	Civil engineering
Temperature avg	Degrees C	V	O	I	I			
Temperature low	Degrees C	V	O		I			
Temperature-high	Degrees C	V	O		I			
Rainfall	Inches	V	O	I	I			
Cloud cover	Percent	V	O					
Insolation	Joules	V	O	I	I			
Res volume	Liters	S		O				
Res withdraw	Liters	V		I	O			
Food crops	Tons	S			O			
Population total	Count	S				O	I	I
Population child	Count	S				O		
Population adult	Count	S				O	I	
Population elder	Count	S				O		
Income average	\$/house	V					O	I
Employment rate	Percent	V					O	O
Paved roads	Km	S						O
Dirt roads	Km	S						

uncertainty of the value. The variable type column indicates whether the value is a state variable (decreased/increased each step) or a simple variable that is calculated at each time step. An “O” in the table is used to indicate that the value is an output of the submodel directly above it and an “I” indicates that the value is an input. The table is not completed until every variable not associated with an “I” is deleted and every other variable is associated with one—and only one—“O.”

2.3.3.3 Model Identification Requirements

As a special case of the submodel requirements identification, model state variables must be initialized with a system state at time step 0 (zero). Depending on the approach to state variable simulation, external data sources such as raster and vector maps, site description tables, entity state descriptions, and external model output (e.g., a global climate model) will be required to seed the model’s state variables. This effort can be as time-consuming and difficult as the development of the model rules and equations. Team members will be assigned to this effort and will similarly debate and establish working contracts with the submodel teams, as described in the previous paragraph.

2.3.4 Construct a Full Dummy Model

To facilitate the integration of submodels into a functioning whole, it is very useful to assemble a “dummy” model that contains simplified versions of all submodels. At a minimum at this stage, each submodel should feed static values of all variables that it is responsible for providing to other submodels. The dummy model serves as a sort of workbench for each submodel development team. Some useful dynamics may be built in as data are available. For example, a dummy weather model might report a fixed series of monthly average temperatures, rainfall, etc., for use elsewhere in the model; or a population model might generate a monotonically increasing total population count needed by another submodel. The goal is to provide a simple, clear model context within which submodel development can be accomplished. That context includes the “wiring” between all submodels, which allows each submodel development team to replace their dummy submodel with working versions while maintaining all expected submodel outputs and using all preestablished inputs.

As submodel development proceeds, developers are likely to discover that they need additional inputs not previously agreed to, or cannot generate certain agreed-upon outputs, or can easily provide potentially useful new outputs not previously discussed. Therefore, during development it is important for teams to communicate all revisions needed to the contracts captured in the “agreement table,” and to generate updated dummy submodel components as appropriate.

2.3.5 *Construct the Submodels*

By this point the submodels have been identified; submodel development teams have been formed; submodel interaction requirements, expectations, and initialization requirements have been documented; and timeframes for completion of the submodel components have been set. Submodel development teams can now focus independently on the further design, refinement, debugging, and sensitivity analysis of the submodel components. If the identified submodels are found to be too unwieldy, they must be broken into manageable components using the procedure described above for decomposing the full model into submodels. A more ambitious model will require several levels of partitioning before it is suitable for individuals to focus on developing the cause-effect mathematics that drive the model.

There is no universal method for constructing submodels because specific objectives and individual modeler capabilities vary to a great degree from project to project. This stage of model development is where individuals can be most creative with respect to the modeling process. Nevertheless, a number of activities and objectives must be considered during a submodel design and development exercise. These may be grouped into two categories: general modeling and group modeling. The general modeling category involves such considerations as keeping the model simple, making sure it can be understood by the intended audience, ensuring that units used in the model conform with the submodel team agreement table, and performing sensitivity analyses. Of more interest here are group modeling principles related to submodel design and development:

- The submodel must be developed within the parameters established for the project.
- Duplication of names for variables, stock, and other values that will be publicly visible (e.g., “age”) must be avoided among teams in order to prevent subsequent difficulties during model integration.
- All modelers must use only the software and hardware designated for the project.
- Submodel development will depend only on inputs generated by other submodels.
- All outputs required from the model must actually be generated by simulations.
- All inputs and outputs are used and generated using the units previously agreed upon by the entire group.
- Submodel development is completed within the negotiated timeframes.
- All required changes are communicated quickly and diplomatically to other submodel teams.
- Submodel teams continue to monitor the internal state and external input variables for their submodel to determine whether the submodel is operating within reasonable parameters.
- Submodels are tested and initially refined using group-generated artificial time series data.

As the individual submodels are completed, they are ready to be integrated into the full model.

2.4 Integrate the Model

When multiple submodels are completed and ready to be joined, the process of integration is usually performed by a subset of the entire modeling team, but that subset should include a representative from each submodel team. Integration is accomplished by inserting finished submodels into the dummy model, one at a time, while testing and debugging them to identify and eliminate errors in program execution and output. Because “interesting” things are likely to happen when finished submodels start working with each other’s actual output for the first time (instead of dummy outputs), the necessary problem-solving work may become vexing if it is not approached systematically.

The most effective general approach to testing and debugging is to add the finished submodels one at a time, first ensuring that NetLogo can execute them without problems, then testing one submodel’s operation in tandem with another one. This approach is familiar to anyone who has tried to diagnose and rectify problems with a complex device or system—a desktop computer or an automobile, for example. The intent is to isolate and resolve one problem at a time in order to avoid complicating the diagnosis process with divided attention or irrelevant variables. Once the two submodels are executing correctly in NetLogo and producing output that is reasonable and realistic, then a third submodel may be inserted and tested.

Although submodel integration must proceed in a controlled, systematic way, it cannot be prescribed as a linear procedure. Integration is an iterative process. It may require repeated testing and debugging work with different combinations of submodels, or even taking a step back to fix a new problem with a previously operational submodel that appeared after debugging a different one. The successful isolation of problems involves detailed inspection of each submodel’s behavior within the context of the evolving operational model, and debugging requires participation by representatives of all submodel teams. The difficulty of these tasks depends on the specific model and the complexity of the system behaviors it simulates. Some trial and error during the debugging and integration of submodels is unavoidable. As noted previously, debugging addresses the two basic issues described below: compilation and logic errors.

2.4.1 *Debug NetLogo Compilation Errors*

Each submodel was developed in relation to the common time series output test environment embodied by the dummy model, so the finished submodel should “plug into” the dummy model with few problems, if any. Nevertheless, when a submodel

is first inserted in place of its corresponding dummy submodel, certain “mechanical” types of errors may emerge right away. We refer to these as *compilation errors*, and they are revealed in the form of NetLogo error messages. They represent obvious errors with respect to the dummy model, such as incorrect unit errors or duplicated variable names that mean different things in two or more submodels. Resolution of these problems is generally a straightforward task.

2.4.2 Debug Errors in Model Logic

As NetLogo compilation problems are resolved and operational testing continues, the submodels will be responding to input combinations not included in the dummy model (i.e., authentic input provided by other completed submodels). At this stage, it is common for test simulations to produce incorrect output or other incoherent behavior. Examples include unexpected cycling, chaotic activity, operation of submodels outside their range of sensitivity, and nonsensical output. These results indicate errors in the logic embodied in the submodels, particularly in terms of how each one interacts with the others. The testing and debugging of model logic draw on the technical expertise of the team and each working group’s deep familiarity with its own submodel. Each team must provide guidance for evaluating whether inputs from other submodels are within the specified ranges. This can take the form of inserting code that produces an explicit error message when a submodel is not receiving valid inputs. These error messages can help to facilitate communication between the submodel teams, but successful debugging further requires that all submodel teams collaborate very closely to assess model integrity and to monitor how their own submodels perform within the context of the whole system. All requirement changes and fixes must be assiduously reflected in the shared model documentation.

2.4.3 Demonstrate to End Users

Once all issues with submodel performance and operation of the integrated model are resolved, the development team provides demonstrations for representative end users. The purpose of the demonstrations is to let the end users run the model and see how it performs when loaded with real or simulated data. Two reasons for providing demonstrations are to observe whether the target users have any unforeseen problems running the model and whether they offer any suggestions for last-minute “tweaks.”

If the modeling team has followed our three-phase development process, no significant problems should emerge. Straightforward changes, such as level of detail displayed in the user interface, may emerge; these are easy to implement. Direct requests by users for more substantive revisions should be considered in terms of

model utility and cost of implementation. Fairly substantive changes are not unheard-of after end user demonstrations, but at the final stage of a project the project team tends to be constrained in terms of revisions it can consider. The nonnegotiable constraint is the project objective: user change requests that exceed the original scope of the project can rarely be satisfied under the original funding and personnel commitments. However, constructive user input for significantly extending the scope or utility of the model should not be discarded: it may suggest the need for a follow-on project when funding and time are available, and may even provide core concepts for a more ambitious model development project in the future.

2.5 Disseminate the Model

Product distribution is not part of the model development process, strictly speaking. But dissemination of the model to a community of interest is usually the final, if unspoken, goal of nonproprietary model builders. Even given the level of interest and effort each member of the multidisciplinary team has invested to fulfill the specific modeling objective, the final product may have important potential uses not recognized by the team. Transparent, open-source models such as those constructed using NetLogo may be readily adaptable for different applications within any of the disciplines represented by the modeling team. An operational model may serve as a sort of template—or at least as a concrete starting point—for specialized or enhanced follow-on models. Teams that wish to maximize the utility of their modeling effort will think about efficient modes of model distribution through the course of the project. Common modes of model dissemination include formal documentation and publication of the model using the *Overview, Design concepts, and Details* or ODD format (Grimm et al. 2006); posting the model and user documentation to appropriate user-community web sites; and distribution on physical media such as CD-ROMs.

2.6 Conclusion

This chapter has outlined a practical framework within which an interdisciplinary research team can design and develop a large, dynamic, spatially explicit ecological landscape simulation model. This framework, repeatedly used with success with our university students over many years, is intended to promote an effective balance between efforts that need approval by the whole development team and efforts that draw on the imagination, expertise, and motivation that arises through the effort of the individual. Full model conceptualization is performed with respect to end user requirements, as tempered by available resources. That conceptualization governs the partitioning of the model into components that can be developed through individual efforts. As the model components are completed within the overall design

requirements, they are linked together as an operational final model. Full model debugging requires modification to the components by the original developers of those components in coordination with the developers of the other components.

Quality leadership in this multidisciplinary environment is crucial for success, and it must be executed with consideration and respect for differences in the personality, background, motivation, and time availability of the team members. Using the approach outlined in this document will help the team leader and members to successfully design, develop, and operate large, complex spatial models.

References

- Grimm V, Berger U, Bastiansen F, Eliassen S, Ginot V, Giske J, Goss-Custard J, Grand T, Heinz SK, Huse G, Huth A, Jepsen JU, Jørgensen C, Mooij WM, Müller B, Pe'er G, Piou C, Railsback SF, Robbins AM, Robbins MM, Rossmanith E, Rüger N, Strand E, Souissi S, Stillman RA, Vabø R, Visser U, DeAngelis DL (2006) A standard protocol for describing individual-based and agent-based models. *Ecol Model* 198(1–2):115–126
- Tomlin D (1990) *Geographic information systems and cartographic modeling*. Prentice-Hall, Englewood Cliffs
- Wilensky U (1999) *NetLogo*. Computer software. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston. <http://ccl.northwestern.edu/netlogo/>. Accessed 01/2011

Ecologist-Developed Spatially-Explicit Dynamic
Landscape Models

Westervelt, J.D.; Cohen, G.L. (Eds.)

2012, XX, 260 p. With online files/update., Hardcover

ISBN: 978-1-4614-1256-4