2

THE MODELING PROCESS WITH SYSTEM DYNAMICS

This chapter provides a foundation for undertaking modeling projects. It describes an end-to-end modeling process (or life cycle) using the technique of system dynamics. Modeling concepts are elaborated with working examples progressively introduced during the steps. Besides "how-to" information, important material on general system behaviors and structural patterns are also presented, which is particularly useful for both developing and understanding modeling results. If you are already familiar with system modeling and/or your purpose is to quickly learn the practical aspects of coding executable models, start with Section 2.6, Model Formulation.

The purpose of modeling is to be better informed about the dynamic consequences of process and policy decisions. This chapter will help illustrate that the scope and process of modeling is driven by the purpose. It contains content regarding general modeling principles drawn from the classic sources [Richardson, Pugh 1981] and [Forrester 1968]. Some additional modeling guidelines come from [Richmond et al. 1990] and various publications from the annual ProSim workshops. Another good reference on the modeling process with system dynamics is [Sterman 2000], which is oriented towards business applications but the same principles hold.

Fortunately, many modeling concepts are similar to those used in software development. Process modeling and simulation is a form of software development after all, and this description of the process is tailored to the software engineering professional in terms of the assumed skillset and application examples used. The software engineer is particularly well suited for applying system dynamics to his/her domain compared to practically all other disciplines.

The building of a model essentially involves defining the requirements for and developing a software application. A systems engineer may be more trained at system conceptualization, but a software engineer has the immediate skills to implement complex models. Many activities and applicable heuristics used in system dynamics modeling are identical or very similar to those used in software development. Some concerns are unique to simulation projects, such as during model validation.

If the modeling process is done well, some major benefits achieved include:

- Development of stakeholder consensus and client ownership, leading to shared commitment to decision making
- Becoming a learning organization
- Better decision making based on improved information from the models
- Improved project execution with lessened risk

Next is a brief overview of the system dynamics technique. After this section is a description of general system behaviors to gain an understanding of time patterns with elemental structures that produce them. Following that are detailed descriptions of modeling activities and model elements. The latter part of the chapter addresses important considerations for software metrics, management guidelines for modeling projects, and tool discussions.

2.1 SYSTEM DYNAMICS BACKGROUND

System dynamics refers to the simulation methodology pioneered by Jay Forrester, which was developed to model complex continuous systems for improving management policies and organizational structures [Forrester 1961, Forrester 1968]. Improvement comes from model-based understandings. This paradigm has been applied to managerial systems for many years, but only recently has software engineering started realizing its potential. This may be due to the fact that the field is still young, and much effort to date has gone into first codifying the software engineering field. Now that many aspects are better defined, it is easier to delve into the processes, model them, and explore alternatives.

System dynamics provides a very rich modeling environment. It can incorporate many formulations including equations, graphs, and tabular data. Models are formulated using continuous quantities interconnected in loops of information feedback and circular causality. The quantities are expressed as levels (also called stocks or accumulations), rates (also called flows), and information links representing the feedback loops.

The system dynamics approach involves the following concepts [Richardson 1991]:

- Defining problems dynamically, in terms of graphs over time
- Striving for an endogenous ("caused within") behavioral view of the significant dynamics of a system

- Thinking of all real systems concepts as continuous quantities interconnected in information feedback loops with circular causality
- Identifying independent levels in the system and their inflow and outflow rates
- Formulating a model capable of reproducing the dynamic problem of concern by itself
- Deriving understandings and applicable policy insights from the resulting model
- Ultimately implementing changes resulting from model-based understandings and insights, which was Forrester's overall goal

A major principle is that the dynamic behavior of a system is a consequence of its own structure. Given this, the structure of a system can be focused on in order to effect different behavior. Improvement of a process thus entails an understanding and modification of its structure. The structures of the as-is and to-be processes are represented in models.

The existence of process feedback is another underlying principle. Elements of a system dynamics model can interact through feedback loops, where a change in one variable affects other variables over time, which in turn affect the original variable. Understanding and taking advantage of feedback effects can provide high leverage.

2.1.1 Conserved Flows Versus Nonconserved Information

In the system dynamics worldview, process entities are represented as aggregated flows over time. The units of these rates are entities flowing per unit of time. These flows are considered material or physical flows that must be conserved within a flow chain. Information connections are not conserved flows, on the other hand. Although the artifacts of software development are information in one form or another, they constitute physical entities in system dynamics models and should not be confused with nonconserved information links.

Information links only provide data from auxiliaries or levels to rates or other auxiliaries. Nothing is lost or gained in the transfer. So whereas software tasks are conserved as they transform through life-cycle phases, information on them is not conserved such as a derived indicator like percent of job complete.

A physical example of a real-world level/rate system is a water network, such as a set of holding tanks connected with valved pipes. It is easy to visualize the rise and fall of water levels in the tanks as inflow and outflow rates are varied. The amount of water in the system is conserved within all the reservoirs and readings of current tank levels are signals that can be propagated anywhere.

2.1.2 The Continuous View Versus Discrete Event Modeling

The continuous view does not track individual events; rather, tasks are treated "in the aggregate" and systems can be described through differential equations. There is a sort of blurring effect on discrete events. The focus is not on specific individuals or events, but instead on the patterns of behavior and on average individuals in a population. System dynamics as a simplifying technique does not lose important resolution, since the large number of individual "events" in a typical software project precludes management overview at that granularity level anyway. When called for, this resolution can be handled in system dynamics using a hybrid approach.

Discrete-event approaches model each and every event. Their focus is usually on the flow of discrete entities without having feedback connections and the resulting internal dynamics. These effects can sometimes be implemented with discrete modeling but, generally, discrete simulation packages do not easily support feedback connections. A continuous representation is mechanically easier to achieve, and it is always possible to transform a continuous model into a discrete one.

Before choosing a modeling method for software processes, an important question that should be answered is "What in the software process is continuous and what is truly discrete?" Are there discrete aspects that need to be preserved for the purpose of the study? If so, then a discrete or hybrid modeling approach may be better suited than system dynamics. See later chapters for more discussions of continuous modeling versus discrete modeling; Appendix B also summarizes some hybrid software process modeling work.

2.1.3 Model Elements and Notations

This section expands on the Chapter 1 summary of model elements. They are summarized along with their notations in Table 2-1. These notations are the ones available in the Ithink and Stella toolsets used for almost all of the examples in this book. Notations used for other popular toolsets are nearly identical and shown in Section 2.13.

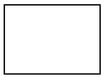
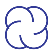
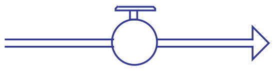
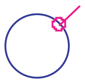
The descriptions in Table 2-1 are elaborated with more detail in later parts of this chapter, especially in section 2.6 Model Formulation and Construction. Figure 2.1, Figure 2.2 and Figure 2.3, respectively, show simplified examples of levels, rates, and auxiliaries with their next-door neighbors for some typical software process structures. Connections to other model elements from the neighbors are not shown for overall legibility.

Elements can be combined together to form infrastructures. Using common existing infrastructures in models can save a lot of time and headache. Two examples of common infrastructures are shown in Figure 2.4. The first is a simple software production infrastructure and the other is a defect propagation infrastructure. Connections to other model elements are not shown for simplification. An infrastructure can be easily modified or enhanced for different modeling purposes. See Chapter 3 for applied examples of software process infrastructures.

2.1.4 Mathematical Formulation of System Dynamics

The mathematics in this section is for general background, and is not necessary for developing or using system dynamics models. It illustrates the underpinnings of the modeling approach and provides a mathematical framework for it. An elegant aspect of system dynamics tools is that systems can be described visually to a large degree, and there is no need for the user to explicitly write or compute differential equations. The tools do all numerical integration calculations. Users do, however, need to compose

Table 2-1. System dynamics model elements

Element	Notation	Description
Level		A level is an accumulation over time, also called a stock or state variable. It can serve as a storage device for material, energy, or information. Contents move through levels via inflow and outflow rates. Levels represent the state variables in a system and are a function of past accumulation of rates. Some examples are: • Software tasks (function points, SLOC, use cases, modules, COTS components, etc.) • Defect levels • Personnel • Expended effort
Source/Sink		Sources and sinks indicate that flows come from or go to somewhere external to the process. Their presence signifies that real-world accumulations occur outside the boundary of the modeled system. They represent infinite supplies or repositories that are not specified in the model. Examples include: • Source of requirements • Software delivered to customers • Employee hiring sources and attrition sinks
Rate		Rates are also called flows; they are the "actions" in a system. They effect the changes in levels. Rates may represent decisions or policy statements. Rates are computed as a function of levels, constants, and auxiliaries. Examples include: • Software productivity rate • Defect generation • Personnel hiring and deallocation • Learning rate • Financial burn rate
Auxiliary		Auxiliaries are converters of input to output, and help elaborate the detail of stock and flow structures. An auxiliary variable must lie in an information link that connects a level to a rate. Auxiliaries often represent "score-keeping" variables. Example variables include: • Percent of job completion • Quantitative goals or planned values • Constants like average delay times • Defect density

(*continued*)

Table 2-1. System dynamics model elements *(continued)*

Element	Notation	Description
Information Link		Information linkages are used to represent information flow (as opposed to material flow). Rates, as control mechanisms, often require connectors from other variables (usually levels or auxiliaries) for decision making. Links can represent closed-path feedback loops between elements. Examples of such information include: • Progress and status information for decision making • Knowledge of defect levels to allocate rework resources • Linking process parameters to rates and other variables.

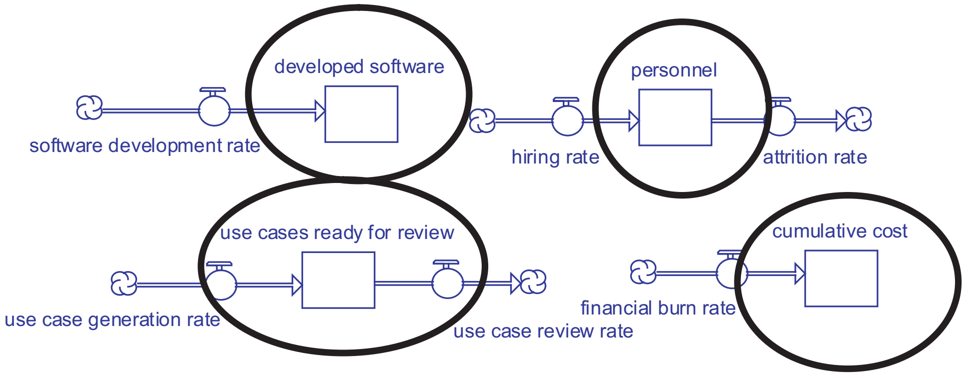


Figure 2.1. Level examples.

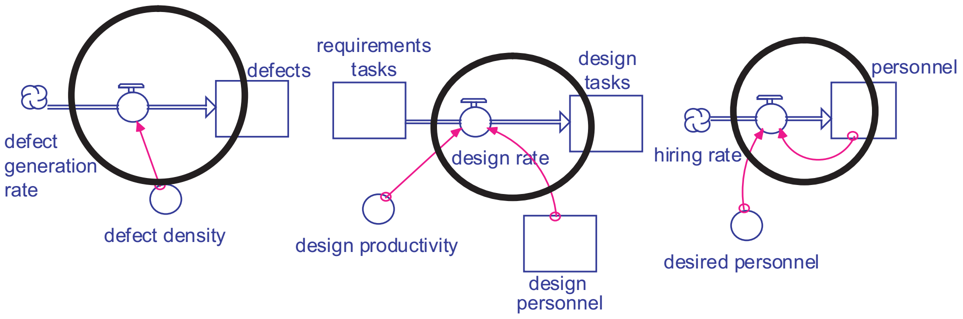


Figure 2.2. Rate examples.

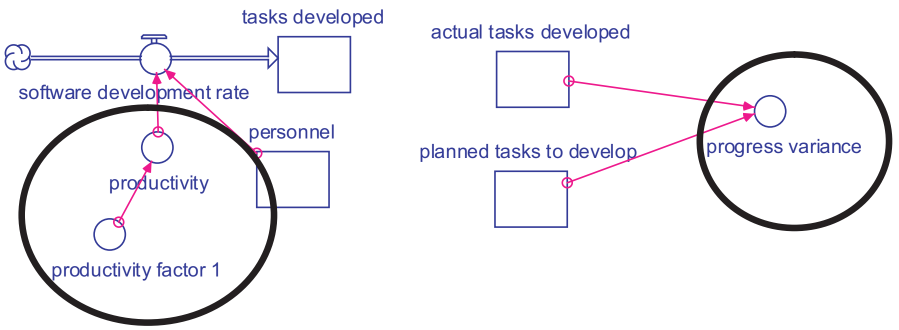


Figure 2.3. Auxiliary examples.

equations for rates and auxiliaries, which can sometimes be described through visual graph relationships.

The mathematical structure of a system dynamics simulation model is a set of coupled, nonlinear, first-order differential equations:

$$\mathbf{x}'(t) = \mathbf{f}(\mathbf{x},\mathbf{p})$$

where **x** is a vector of levels, **p** a set of parameters and **f** is a nonlinear vector-valued function. State variables are represented by the levels. As simulation time advances, all rates are evaluated and integrated to compute the current levels. Runge–Kutta or Euler's numerical integration methods are normally used. These algorithms are described in standard references on numerical analysis methods or provided in technical documentation with system dynamics modeling tools such as [Richmond et al. 1990].

Numerical integration in system dynamics implements the following calculus integral for determining levels at any time *t* based on their inflow and outflow rates:

$$Level = Level_0 + \int_0^t (\text{inflow} - \text{outflow})\, dt$$

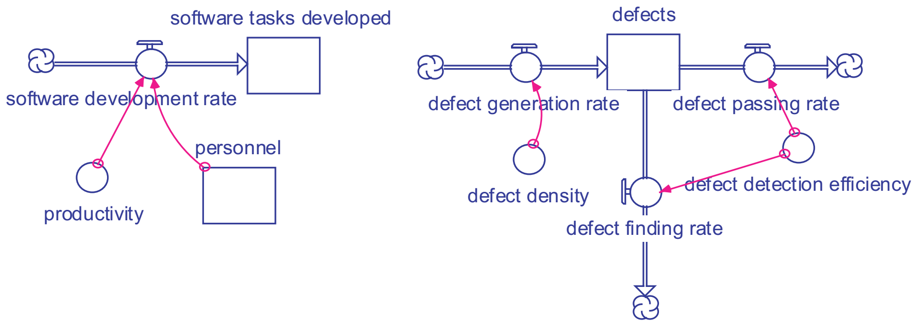


Figure 2.4. Infrastructure examples.

The *dt* parameter corresponds to the chosen time increment for execution. Corresponding system dynamics code for the level calculations would be:

$$Level(\text{time}) = Level(\text{time} - dt) + (inflow - outflow) \cdot dt$$

$$\text{INIT } Level = Level_0$$

where *Level* is computed for any time, $Level_0$ is the initial level value, and the flow rates to and from the level are *inflow* and *outflow*, respectively. Describing the system with equations like the above spares the modeler from integration mechanics. Note that almost all tools also relieve the modeler from constructing the equations; rather, a diagrammatic representation is drawn and the underlying equations are automatically produced. See the following sections for more detail of the model components.

2.1.5 Using Heuristics

One of the ultimate challenges in modeling is dealing with the complexity of the systems under study. Since the systems being modeled are complex, the task of modeling them can also become complex without some simplifying guidelines. Both art and science are called for. Model conceptualization in particular cannot be algorithmically described. It is akin to architecting complex systems that exhibit ambiguity and unboundedness. Dr. Eberhardt Rechtin elucidates the heuristic process used by architects of complex systems in the insightful and entertaining book, *Systems Architecting* [Rechtin 1991], and the follow-on, *The Art of Systems Architecting* [Rechtin, Maier 1997]. Many of the heuristics also apply in simulation modeling.

Heuristics are normally described with short aphorisms. They are used as guidelines or rules of thumb to help simplify the multivariate decision process in the midst of complexity. They help to reduce the search for solutions based on contextual information. Relevant heuristics to deal with the complexities and help simplify the modeling are identified throughout this chapter. An example is,

Don’t try to model the “system.”

This heuristic helps reduce modeling complexity by not trying to include everything in a model. It reminds one that a model addresses a particular problem and that irrelevant details should be abstracted out. The heuristics serve as general guidelines for dealing with hard modeling issues and are summarized at the end of this chapter in Section 2.15.1, Summary of Modeling Heuristics.

2.1.6 Potential Pitfalls

Computer models using system dynamics may on occasion exhibit stability problems, system lags and delays, discrete transients, off-nominal behavior problems, saturation effects, and so on. Sound and rigorous modeling can overcome these problems that ultimately stem from specification inadequacies. Methods to avoid some of these problems are described later in the book, and serious modelers may want to consult other system dynamics implementation references for more detail.

Recall that system dynamics is a technique that models continuous variables, so discrete entities lose their individuality and are "blurred" together. This may represent a new change of perspective for some readers. Keep in mind that models are abstractions of reality, and continuous models will compute fractional quantities of things normally thought of as discrete. The results are just as meaningful for an aggregate view as if the quantities were discretized.

2.2 GENERAL SYSTEM BEHAVIORS

This section will review general system behaviors representative of many types of systems, and illustrate them with simple structures. Knowing how systems behave and respond to given inputs is valuable intuition for the modeler. This knowledge can be used during model assessment, such as when evaluating the system response to test inputs used to stimulate the system behavioral modes.

The *order* of a system refers to the number of levels contained. A single-level system cannot oscillate, but a system with at least two levels can oscillate because one part of the system can be in disequilibrium.

2.2.1 Goal-Seeking Behavior

Figure 2.5 shows generic goal-seeking behaviors found in feedback systems, as modified from [Forrester 1968]. Time starts at the beginning of a project, process improvement program, or other goal-attainment activity. Examples of the goal or measurement being tracked could be product quality as measured by defect density trends or average

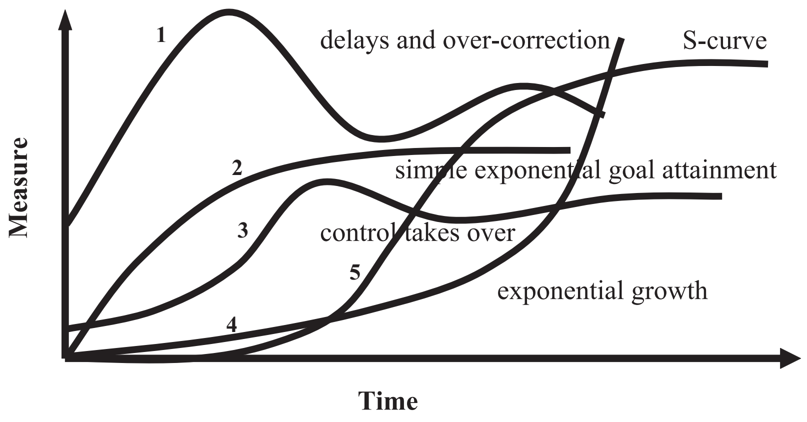


Figure 2.5. Goal-seeking behaviors.

system response time; process performance such as earned value, productivity, or defect yield; organizational hiring; financial goals such as sales; and so on. Rudimentary model structures and inputs that produce these dynamic behaviors are shown in Section 2.2.3.

Curve #1 at the top is the result of excessive delays in the control loop or overcorrections. Consider a shower with sluggish controls that you are getting used to. The water temperature oscillates wildly as you adapt to the internal delays of the temperature control system. In a software process, this would be akin to late metrics feedback. Possibly, signals from the process are not being interpreted properly by management, "an unwise prince who doesn't recognize."

The next curve, #2, is the result of an exponential goal attainment feedback loop (also called negative or balancing feedback). The curve gradually reaches the set point and eventually levels off to the desired quantity. The system reaches equilibrium in a controlled manner. It shows negative feedback for a positive goal. For a zero goal, the curve mirrors itself by approaching from the top and asymptotically approaching the goal. The curve may represent the number of people when hiring toward a desired number. The speed at which the goal is attained depends on the hiring time delay. This will be used as an exercise later to examine the curve sensitivity to varying time delays.

Curve #3 starts out with an early exponential growth pattern until control takes over. From that point, the deviations diminish and the goal is gradually eased into. It is quite possible that a system will not recover and continue to oscillate, such as when there are excessive delays and overcorrecting taking place.

Curve #4 starts at the bottom and is an exponential growth curve that dramatically rises. It is best plotted on logarithmic paper to cover the wide range of values. This type of growth has been observed for defect-fixing costs as the life cycle progresses (whereby fixing overhead accumulates), early growth of new product users or sales, global Internet traffic or online program usage, growth of regional Internet users until the area becomes saturated, early corporate growth, and so on. A market stagnation could cause enterprise-related exponential curves to eventually peak and level off like the third curve when control takes over.

Sigmoidal behavior in curve #5 refers to the shape of a graphed quantity that exhibits a sigmoidal or an "S" shape. This shape is commonly called an S-curve. S-curves are ubiquitous in systems. They are flatter at the beginning and end, and steeper in the middle. The S-curve quantity starts slowly, accelerates, and then tails off. S-shaped progress curves are often seen on projects because cumulative progress starts out slowly, the slope increases as momentum gains, then work tapers off. S-curves are also observed in technology adoption and corporate and sales growth where saturation points are reached (the early portions of these are exponential). The ROI of technology adoption is also S-shaped, whether plotted as a time-based return or as a production function that relates ROI to investment.

Several examples of S-curve structures and behaviors are presented in Chapters 3, 4, 5, and 6, including technology adoption, software virus propagation, sales growth and stagnation, project effort expenditures when staffing approximates a Rayleigh curve, and more.

These curves illustrate that feedback loops must be finely tuned. The right information must be used to effect feedback in the proper manner. A system can go into oscillations if the wrong or poorly timed feedback is used (just like the shower temperature with sluggish controls). Too much positive feedback can produce system oscillation. A system can become overdamped if there is too much negative feedback, because there will be more resistance than is needed.

2.2.2 Information Smoothing

Smoothing of information is used to detect real trends by eliminating random spikes. Decision makers should not interpret a day's jump in a relatively long-term pattern of some quantity as a permanent trend to base decisions on. Therefore, information is averaged over a sufficient time period.

Smoothed variables exponentially seek the input signal. Information smoothing can be modeled as a first-order negative feedback loop. The smoothing curve could represent information on the apparent level of a system as understanding increases toward the true value. There is a time delay associated with the smoothing, and the change toward the final value starts rapidly and decreases as the discrepancy narrows between the present and final values. A graphic example of a smoothed variable that follows an input signal that steps up and down is in Figure 2.6. Example structures and behaviors for information smoothing are shown later in Chapter 3.

2.2.3 Example: Basic Structures for General Behaviors

Now let us observe rudimentary behaviors using simple models. A single rate/level pair will be used to demonstrate the general behaviors. Figure 2.7 shows the rate/level model used for the nonfeedback tests for pulse, step, ramp, and Rayleigh curve rate inputs. The rate represents the net flow in or out of the level. Figure 2.8 shows the simple negative and positive feedback systems for the last two examples in Table 2-2. All of these are first-order systems because only one level is contained in them.

Table 2-2 shows the flow inputs, situations they could model in the software process, and the level response. When the flow type is characterized as a staffing profile, the level corresponds to the cumulative number of staff. When the flow is a defect generation rate, the level represents the cumulative defects, and so on. The positive

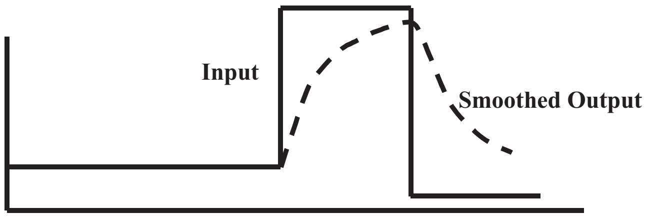


Figure 2.6. Information smoothing.

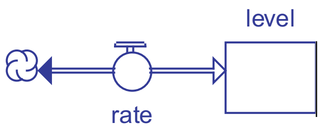


Figure 2.7. Rate/level system (no feedback).

feedback keeps increasing per the growth fraction to represent exponential growth. The negative feedback rate diminishes over time as the level reaches its goal of five (e.g., hiring, quality, or process maturity goals).

Note how a pulse input produces a step increase in the corresponding level. A step input produces a ramp, and a ramp input creates a nonlinear rise in the level. The Rayleigh curve input produces an S-shaped response in the level.

The graph pad format in Figure 2.9 is a handy worksheet for plotting rates and levels manually. It shows the same case as the step input response in Table 2-2. One should be able to calculate a level from rate inputs and, conversely, derive the rate that must have existed to produce indicated changes to a level (at least for linear trends). The units should be derivable also. The interested student should practice this and develop the skill for integrating simple rates. Though simulation software does this for you, it is extremely valuable for modelers to crosscheck and be able to determine expected responses.

2.3 MODELING OVERVIEW

This section contains an overview of the modeling process and lists the basic steps. Much of this section is derived from [Richardson, Pugh 1981] and [Richmond 1994]. The [Richardson, Pugh 1981] reference represents classic, traditional system dynamics in the pre-Graphical User Interface (GUI) Dynamo era, whereas Richmond pioneered visual system dynamics starting with the very early Macintosh models. The differing historical perspectives likely explain the additional manual steps from conceptualization to final model elaboration (such as using causal loop diagrams) in the older approach.

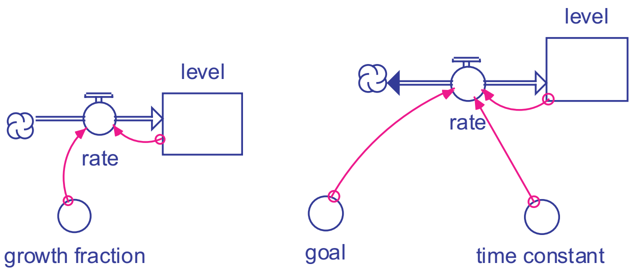
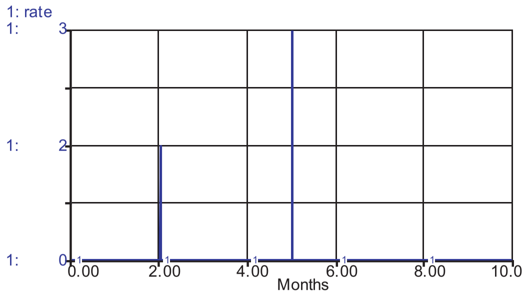
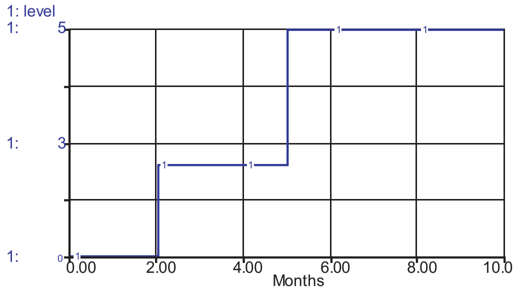
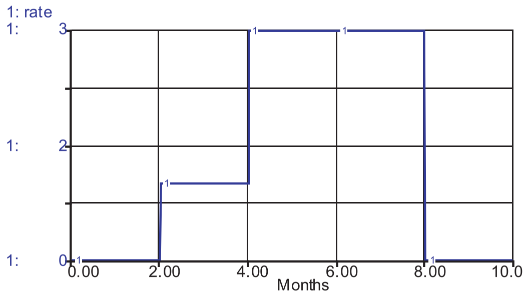
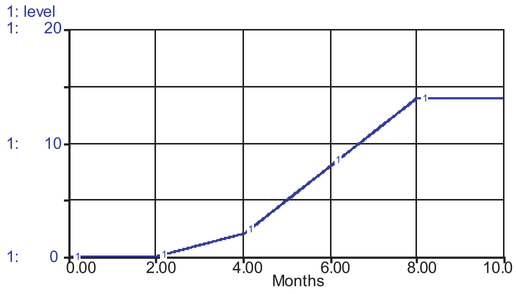


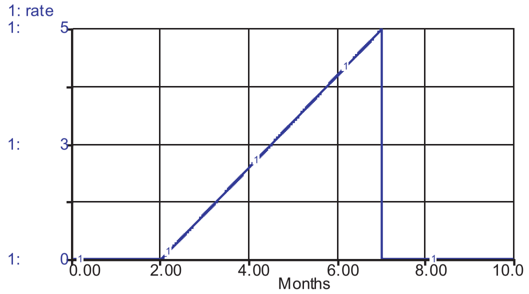
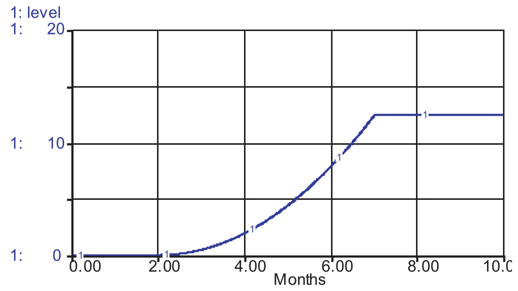
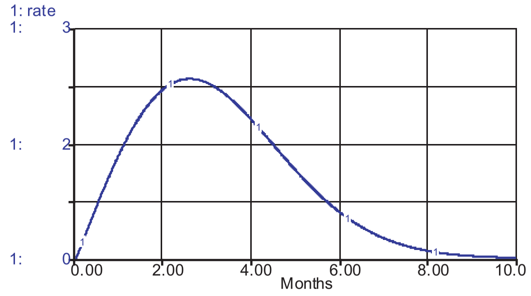
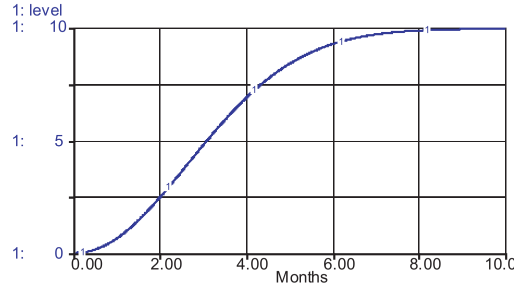
Figure 2.8. Positive and negative feedback systems.

Table 2-2. General behaviors from model structures

Flow Input Type	Software Process Examples	Net Flow Rate	Level
Pulse	• New staff additions • Requirements changes • New portfolio projects		
Step	• Staffing rate • Requirements generation • Defect generation		

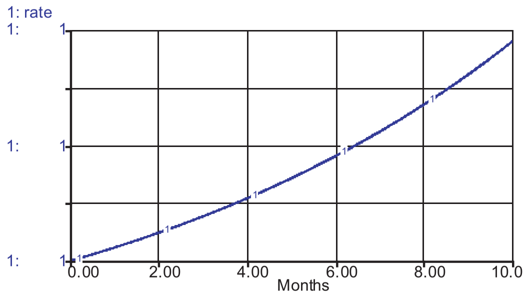
(continued)

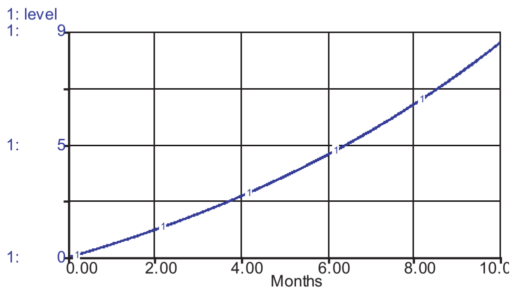
Table 2-2. General behaviors from model structures *(continued)*

Flow Input Type	Software Process Examples	Net Flow Rate	Level
Ramp	• Defect generation before unrealistic deadline • Software check-in rate before build deadline • User growth		
Rayleigh Curve	Project effort expenditure • Defect generation • Technology adoption		

Positive Feedback

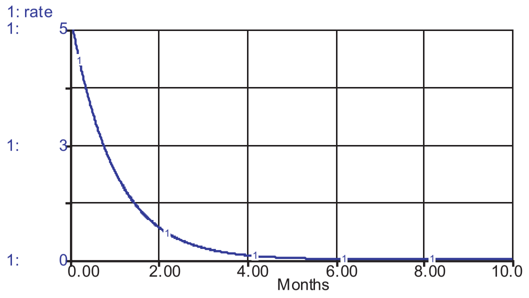
- Software entropy
- Users
- Market size

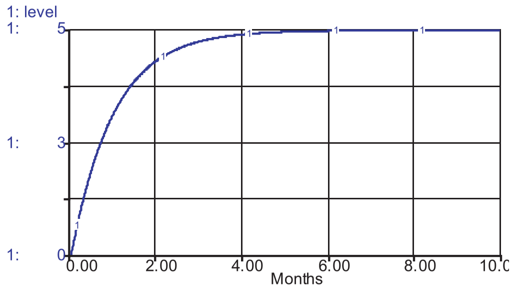




Negative Feedback

- Hiring
- Quality goals
- Process maturity goals





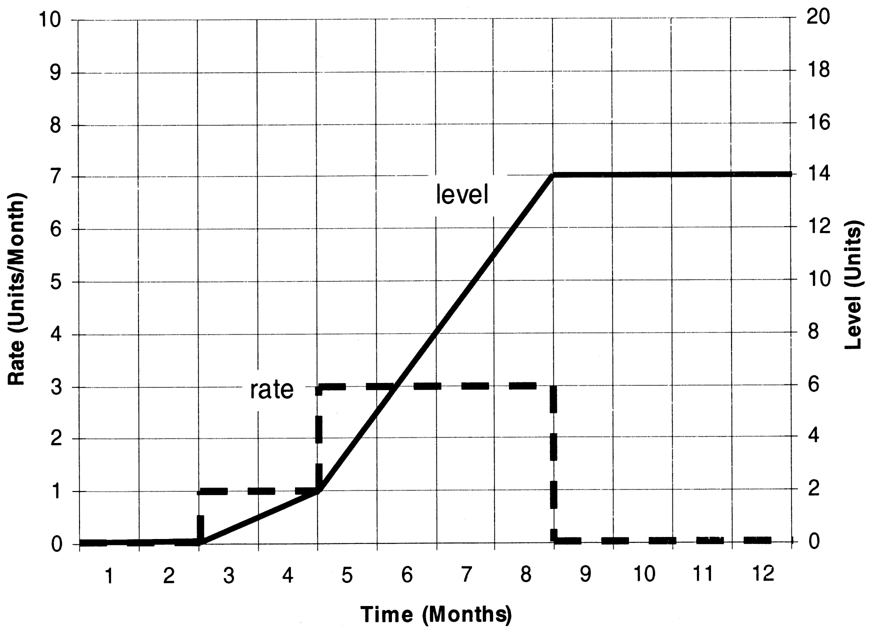


Figure 2.9. Manual graph depiction for step input.

In the early stages of problem identification and model conceptualization, the activities include developing a statement of the problem context and symptoms, articulating the modeling purpose, identifying and drawing reference behavior modes, defining a system boundary, and beginning the system structure of information feedback loops. The purpose will drive the modeling process and scope.

2.3.1 An Iterative Process

Modeling with system dynamics is an inherently iterative process. The complexities of the problems being addressed by modeling tend to force an iterative approach. One passes through several sequences of conceptualization, formulation, simulation, validation, and reconceptualization as a model is elaborated and refined over time. Even if a model can be fully specified at once, it is highly recommended to assemble it in pieces and test the model as it continues to be integrated.

The process is also iterative in that one does not complete a step, consider it finished, then set it aside and move onto the next step. You arrive at an interim completion of the first step, and during the next step you return to clarify the work done in the first step. Revisiting your previous work repeats throughout the modeling process.

The iterative software lifecycle process overviewed in Chapter 1 is also an appropriate general framework for modeling projects. Variants of the system dynamics modeling process have always been iterative and cyclic, similar to evolutionary and nonsequential software development. A mapping of the generic phases used in traditional software development processes (as described in Chapter 1) to the specific modeling steps is shown in Table 2-3.

The process shown in Figure 2.10 is adapted from [Richardson, Pugh 1981]. It starts with problem definition and progresses clockwise through the phases. System understanding can be enhanced at different stages of the modeling process (indicated by the nonsequential connections back to system undertstandings), and the improved understanding helps further the modeling. The overall process may be repeated as software policies improve.

Figure 2.11 shows the modeling stages and their respective concerns adapted from [Richardson, Pugh 1981]. There is a large overlap of concerns between the stages. This again indicates some of the nonsequential aspects of modeling. Minicycles may occur at anytime, and cycles can be going on in parallel, as in the spiral model for software development.

The process has distinct similarities to the spiral model of software development with WinWin extensions. Both start with an evolutionary premise and rely on successive improvement cycles interlaced with mission redefinition based on usage of the preceding product. Defining the model purpose is essentially the same as defining stakeholders and their win conditions, and should be considered just as critical to modeling success. The product of a system dynamics analysis could be the resulting knowledge (estimate, new insight, process improvement action item) gleaned from a working model as well as the model (tool) itself.

Figure 2.12 shows a first-order mapping between the WinWin spiral model and the traditional modeling phases. System understanding is not shown because it is continuous throughout. The inner core shows the WinWin spiral model and the outer layer shows the process modeling activities. Realizing that this is a simplification, there may be additional cycles between some of the steps to reduce risk (such as going from concept to model formulation). There is a high degree of correlation between the life-cycle models, particularly when one explores the details of the steps.

The similarities indicate that the iterative spiral model is a good fit to simulation model development in addition to other types of software. Some of the activities for modeling are unique, of course, such as the extensive nature of model validation. Additionally, simulation models frequently have a smaller user base and time horizon compared to other software. They are sometimes used just long enough to determine that a process improvement is necessary, and they become outdated or irrelevant once

Table 2-3. Mapping of software life-cycle process phases to modeling steps

Generic Software Life-cycle Phase	Modeling Steps
Inception	Problem Definition
Elaboration	Model Conceptualization
	Top-Level Model Formulation
Construction	Detailed Model Formulation and Construction
	Simulation
	Model Assessment
Transition	Policy Analysis and Implementation

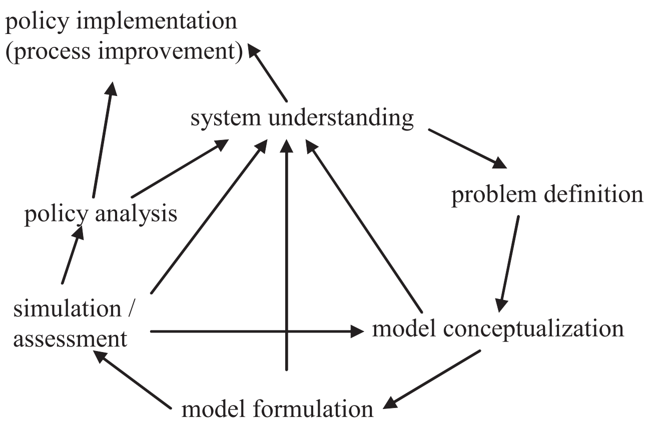


Figure 2.10. Cyclic modeling process.

the process is changed. Sometimes they should still be created to be easily evolvable; it all depends on the intended purposes.

2.3.2 Applying the WinWin Spiral Model

An executable software process dynamics model is a software artifact. Not too surprisingly, then, we have found that good strategic software development processes such as the WinWin Spiral model used by MBASE work well in developing process models.

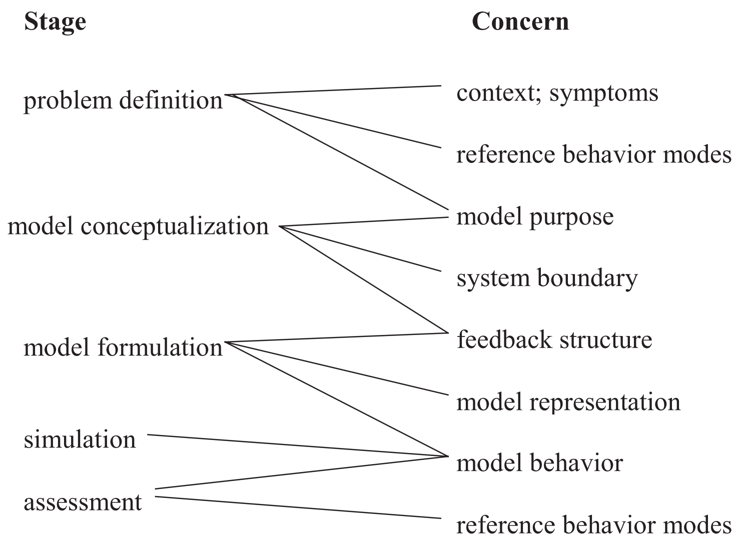


Figure 2.11. Modeling stages and concerns.

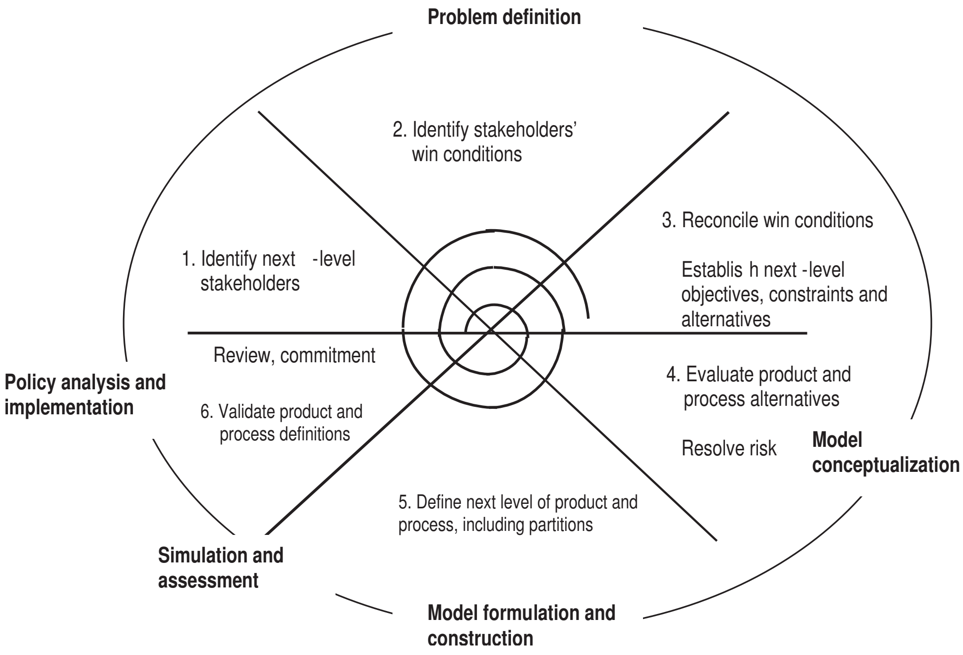

Problem definition
2. Identify stakeholders' win conditions
3. Reconcile win conditions
Establis h next -level objectives, constraints and alternatives
1. Identify next -level stakeholders
Review, commitment
Policy analysis and implementation
6. Validate product and process definitions
4. Evaluate product and process alternatives
Resolve risk
Model conceptualization
5. Define next level of product and process, including partitions
Simulation and assessment
Model formulation and construction

Figure 2.12. WinWin spiral model and process modeling phases.

The WinWin Spiral model approach has the following major strategic steps and guidelines when applied to the modeling process:

1. Use a stakeholder win–win negotiation process to determine the objectives, constraints, and alternative solution approaches for your model.
2. Evaluate the alternative modeling choices for their ability to satisfy the objectives within the constraints.
3. Identify risk areas (e.g., model performance, calibration data, reusability of existing models) in which there is too much uncertainty to evaluate and decide on the choice of model.
4. Buy information to reduce the risk of making the wrong decision via model prototyping, benchmarking, reference checking, or other analyses.
5. Choose the best alternative, or carry two or three along initially if they are different but about equally likely to succeed, based on what you know so far.
6. Within this context, repeat the steps at the next level of detail until your modeling objectives are achieved.

2.3.2.1 Example: Modeling Strategy

Let us see how this works with respect to an example. Suppose you are finding that your normal inspection techniques are not catching as many defects per hour invested when you are inspecting COTS integration glue code, and you would like to model the situation to determine a potentially more cost-effective mix of inspections, benchmarking techniques, and testing techniques that increases your defect detection yield for glue code. Your main alternatives are:

A. Modify an existing systems dynamics model of the inspection and testing process for built-from-scratch software.
B. Develop a new dynamic model that includes the effects of COTS assessment; COTS tailoring; glue code development, inspection and testing; and COTS volatility.
C. Analyze a few glue code inspection results for differences in classes of defects found, and use this to develop a simple, informal nondynamic model of options to best remove late-detected or undetected defects.
D. Perform some combination of options A, B, and C.

Applying the WinWin Spiral Model to this situation might produce the following results:

1. Some of your stakeholders may have significant constraints on the amount of money and time available for the modeling activity. Others may have ambitious objectives for model generality and fidelity.

2., 3. When you evaluate the alternatives with respect to these objectives and constraints, you find that the full form of Alternative B would be an overkill, and that there is insufficient data to assess how much time and effort it would take to modify the existing model (A) to satisfy the stakeholders' generality and fidelity objectives.

4. Analyze a couple of representative glue code inspection and integration processes (a subset of C) to better determine how much modification of the existing model is necessary to achieve various levels of fidelity.

5., 6. If a key stakeholder has a very tight time constraint (e.g., a new-project proposal), choose alternative C to satisfy this near-term need. If other key stakeholders still have strong needs for a dynamic model and model A looks satisfactory to modify, choose to do C, and then do A based on the additional insights gained from doing C. If it is not clear whether A or a subset of B would be preferable, choose to do C followed by another round of the spiral to establish a stronger basis for choosing between A and an appropriate subset of B.

Thus, the WinWin Spiral Model provides strategic guidance with respect to the top-level choices you make, in determining what combinations and sequences of modeling activities to pursue in order to best achieve your stakeholders' objectives. As you elaborate the definition and development of the models, more modeling-specific process guidelines become more important. These modeling steps are now expanded on in the following sections.

2.4 PROBLEM DEFINITION

Problem definition is of primary importance as it sets the stage for subsequent modeling activities. You can make your biggest mistakes in this phase and doom the entire study. At the outset, you define the purpose of the modeling study and its intended audience. Major process stakeholders should be involved from the very beginning to define model purpose. The problem must be important enough to the involved stakeholders to warrant a study. Make explicit what system behavior is of interest and what policies are to be simulated. Consider the type and degree of model implementation; will it be a short exercise that only develops causal diagrams, a several-month model development to test policies, or are you creating an enterprise-wide distributed simulation to be used by many?

Identifying the problem to be addressed requires an unambiguous and explicit definition. The context and symptoms of the problem should be concise and easily verbalized. One informal test of this is called the "elevator test": Can the problem statement be summarized to others well enough to be understood in a minute-long elevator ride?

When defining the problem, you make initial attempts to sketch out system structures and develop reference behavior patterns. These reference behaviors help define the problem dynamically in terms of plots over time. The reference modes can show the problem behavior, desirable to-be system behavior, or observed behavior as a result of previous process policy changes.

2.4.1 Defining the Purpose

A model is created to answer a specific set of questions relating to organizational goals. Defining the goals and deriving questions related to the goals correspond to the first two steps of the Goal–Question–Metric paradigm [Basili 1992], while specific model metrics answering the questions are identified later. The modeling effort strives to improve understanding of the relationships between feedback structure and dynamic system behavior. With this understanding, policies can be developed to improve future process behavior. In some research contexts, a model may be developed to test a theory, but the focus is primarily on modeling for process design and improvement.

The model purpose statement includes identifying the stakeholder audience, the desired set of policies to be simulated, and the type of model implementation. The objectives must be realistic and credible. The degree of implementation can vary across the spectrum. Examples of progressive levels of implementation include:

- Basic consciousness raising via causal loop diagramming, with no executable models developed
- A short modeling exercise to compare policies for an upcoming project decision (like a one-off Brooks's Law analysis)
- Actual adoption of new policies tested by a model and continued policy analysis with the model
- Developing institutional models with dozens of factors for ongoing project estimation and planning

Consider whether the effort will be a one-time adoption of new policies or an ongoing process for continual analysis. The level of implementation clearly will have an important effect on the resulting model content.

Defining the purpose thus includes early consideration of acceptance criteria and process transition planning. Implementation of eventual process changes must be consciously planned from the start. The ability to adapt to change in the organization must be accounted for. The possible side impacts on related processes or other systems should be assessed. Consider what sort of training or reeducation of process participants might be necessary.

Following is a ditty from [Richardson, Pugh 1981] that uses analogy to stress that you need a clear modeling purpose to know where you are going:

A model without a purpose is like a ship without a sail,
A boat without a rudder, a hammer without a nail . . .

Write the purpose down, and be specific. Instead of beginning a study with a top-level goal like "analyze inspections," refine it to something specific and measurable like "What is the dynamic cost profile over time of performing inspections?" or "What is the optimal staffing policy for inspection rework?" Do not try to "model the system" as a goal. A clear, operational purpose is needed or the resulting model will be too big and complex because everything will be included.

The purpose statement should be supplemented with specific behaviors or patterns. These reference behaviors are expressed as graphs of important system variables over time, and described in the next section.

2.4.2 Reference Behavior

Reference behavior patterns help define a problem dynamically; they focus model conceptualization and are used in validation. They are a plot over time of a few variables that best characterize the dynamic phenomenon to be understood. These graphs may include observed and/or desired patterns. They make the purpose statement operational as a graph. They also help specify the time horizon for the model.

The resulting model is tested in subsequent stages of development to see if the reference modes can be reproduced. The ability to do so constitutes an acceptance criterion for the model. Even if this fails, things may be learned that still contribute to an overall successful modeling effort. One possible outcome is that the initial reference modes were not appropriate or misleading.

Reference behaviors are ideally based on actual data. When data is lacking, some of the graphs may need to be inferred and/or drawn on relative scales. Even with good data, it is often advantageous to normalize a variable used in a reference behavior pattern. Absolute quantities may need to be carefully bounded, whereas relative measures are often easier to interpret and are more forgiving since they are valid across a wider range of values, and generally lead to relevant questions about important relationships. Reference behaviors should indicate problematic behavior, not the variables that are causing the behavior.

Not having data for all quantities should not hamper model specification and elaboration. More subjective measures will always be harder to get hard data for, for example, motivation, yet are still important determinants of system behavior. Often, the modeler will have to rely on expert judgment and the intuitions of those closest to the process. Sometimes, other models can be leveraged. Various examples of dealing with missing or incomplete data are described in Chapters 4–7.

2.4.3 Example: Model Purpose and Reference Behavior

Sample goals for the Brooks's Law model may include:

- Understanding the dynamic phenomena behind Brooks's Law
- Improving project execution for a current project
- Improving future project planning

Based on these goals, sample relevant questions to answer are:

- What is the impact on productivity of adding people?
- How much extra communication overhead will result from adding people to a project?
- How much effort will be required to train new people?
- Can people be added to the ongoing project X and can we come out ahead schedule-wise? If so, what is the optimal number of people to add? When is the best time to add them?
- How big should our teams be on future projects?

The implied reference behavior for the Brooks's Law model is an overrunning project. It can be visualized as productivity and personnel over time or planned versus actual progress over time in terms of tasks developed (Figures 2.13 and 2.14).

2.5 MODEL CONCEPTUALIZATION

Conceptualization includes identifying the system boundary to determine what is relevant to the study, and deriving a high-level view of system elements. The boundary must include the parts necessary to generate the behavior of interest, including all relevant policy levers. Some of the process stakeholders should continue their involvement in this stage if they are knowledgeable enough and time should be dedicated to help conceptualize a high-level view of the relevant process(es).

A top-down, iterative approach to model conceptualization generally works best. Compose the model first with small, simple pieces and iteratively broaden it to larger sectors. Build it up using relatively small changes. This way, the gap between iteration versions can be understood and tracked and the model debugged more easily. The same "build a little, test a little" approach also applies during later stages of model formulation.

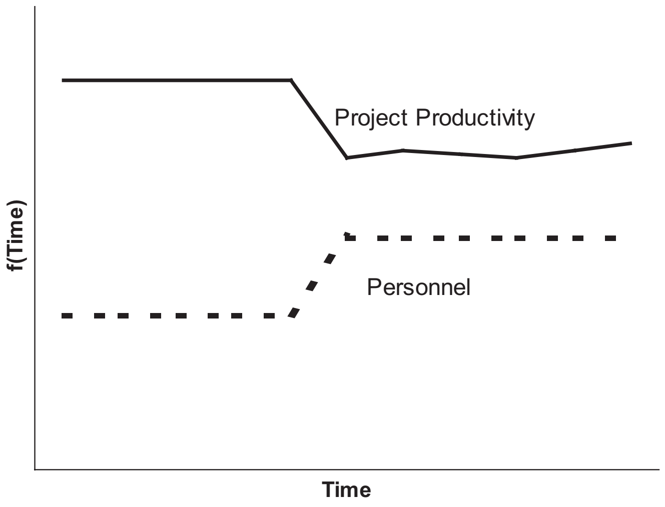


Figure 2.13. Brooks's Law reference behavior—productivity and personnel.

The typical order of conceiving model elements within the boundary is:

1. Develop the physical structure of the system
2. Identify basic information flows
3. Distinguish perceived versus actual information in the system
4. Identify the decisions made by actors in the system resulting from the perceptions

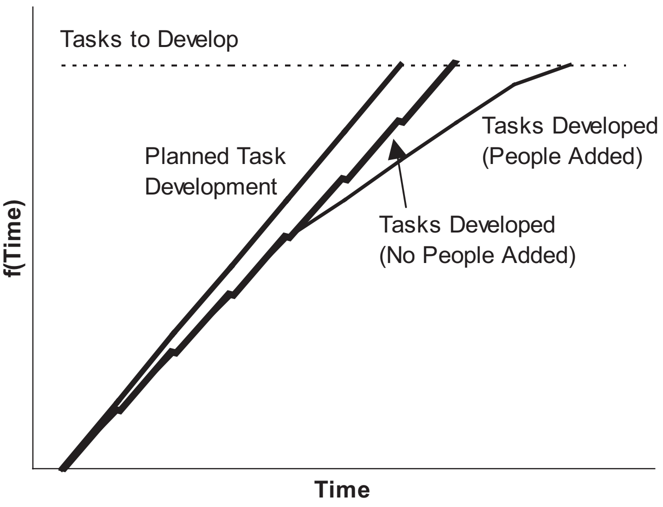


Figure 2.14. Brooks's Law reference behavior—task development.

The perception-based decisions lead to pressures that change the system behavior somehow.

First, develop a system diagram using a high-level depiction of key sectors within a model whose interplay is thought to generate the reference behavior. This focuses on the structural aspects of the system. For software, the physical structure reflects the organization and its processes for developing and transforming software artifacts. Example sectors would be project planning, software construction, and so on. Differentiate physical exchanges (conserved flows) from information connections (that are nonconserved "data links"). Later, the sectors will be elaborated to identify flow chains. At this stage, it is best to concentrate on the system as is, since the desired to-be system will eventually have to be implemented on top of what currently exists anyway.

The decisions represented in the system may consist of the following elements: an actual system state, a perceived state, a desired state, pressures to close the gap and achieve the desired state, and resultant action to change the state and thus close the feedback loop. Often, the delineation of perceived versus actual information gets to the heart of the problematic system behavior. Only that information actually used by actors in the process for decision making should be included. Reflecting this reality, all variables cannot be directly linked in a model if they are not so in the real world.

Early on, a dynamic hypothesis about the feedback structures thought to cause the problem behavior should be attempted. The statement can be verbal or graphic. It may include possible causes of the problem behavior. Keep in mind that such hypotheses are refined throughout the modeling process. It is quite possible that a succinct, well-focused, and consistent hypothesis statement will be realized only after many modeling iterations of conceptualization, formulation, simulation, and evaluation.

One must aggregate and abstract factors to the appropriate degree, though it is a bit of a practiced art to make all model elements relevant to general stakeholders. How much aggregation versus "drilling down" into the process should one undertake? Initially, aggregate things until the level of aggregation obfuscates entities or factors deemed important to the study. Decompose factors only when the problem calls for it.

Always try to match the level of detail to the audience. Represent the system at a level of abstraction that is meaningful to the stakeholders and matches their mindset. Different levels of an organization will expect different degrees of roll-up or data aggregation. Also address the problem statement commensurate with the current stage of model elaboration. The measures of effectiveness in the model and the result variables must be consistent with real-world measures of process performance.

The KISS principle (keep it simple, stupid) should always be kept in mind during model conceptualization and formulation. Do not bite off more than you can chew and strive to minimize the model complexity. The resulting model will be easier to develop and be understood by others.

The following concepts in the Brooks's Law model would be identified at this stage:

- Task flows and personnel flows should be covered
- Effects of adding more people must be included in the formulation
- A schedule completion date must be calculable in the model

2.5.1 Identification of System Boundary

A boundary is the imaginary line delineating what is considered inside and outside the system for modeling purposes. Within the boundary are concepts and quantities thought to be significant to the dynamics of the problem being addressed. Without a relevant boundary, "analysis paralysis" becomes a risk since too many factors keep being considered.

Where exactly does one draw the boundary line between systems and subsystems? Since systems exist on many interacting levels, it is a modeling challenge to cleanly isolate the elements of study. The naturalist John Muir illustrated this conundrum of systems as follows: "When we try to pick out anything by itself, we find it hitched to everything else in the universe."[1]

Defining the boundary of the system is of utmost importance to isolate exactly what parts of the system are necessary to generate the behavior of interest. The boundary must be wide enough to include information feedback loops that help cause system behavior, applicable policy levers, and variables used to evaluate the process in response to new policies. The variables used to assess process changes based on new policies are sometimes called "result variables," as described in Chapter 1.

The policy levers to be included are intervention points that represent leverage in the system, that is, the places where system impacts can be made in the real world and first tested in a model. System components not relevant to the problem should be excluded. When in doubt, recall that the focus is on a problem, not "the system." Otherwise, it is tempting to include too many factors.

It is impossible to define a boundary without a model purpose. When the questions of the modeling study are known, then the importance of factors to include or exclude from the system can be judged. One distinguishes what is explicitly inside, what is implicitly represented inside through aggregation and interpretation of system variables, and what is explicitly outside.

A boundary should be drawn that encloses the smallest number of possible elements. Do not ask whether a piece is in the system or not but, rather, consider whether the behavior of interest will disappear or be erroneously modeled if the piece is excluded. If a component can be easily removed without defeating the purpose, then take it out. It is useful to keep in result variables since they are indicators to assess process performance and measure the relative gains of new or revised policies.

Other questions to consider when defining the boundary are the following:

- What are the physical processes in the system relevant to the problem?
- What are the perceptions of those processes to those in the system and how are those perceptions formed?

[1]John Muir, *My First Summer in the Sierra,* 1911. Interestingly, a connection of natural forces and software processes occurred in Los Angeles in the 1994 Northridge earthquake. Some of us were intimate with software projects that had official schedule reprieves due to facility cleanup and employee situations dealing with serious home damage. The lingering project effects of the earthquake lasted for many months thereafter.

- How do the perceptions help create pressures that influence the physical processes?

Answering these questions will set the stage for model conceptualization.

The physical processes in the Brooks's Law model include software task development (implying a flow chain from required tasks to completed tasks), people coming on board the project, added communication, and training. A perception of project lateness is necessary for a decision to be made to add people, as realized by the amount of software developed to date compared to a desired amount. At some "breaking point," whereby the gap becomes large enough for action, the project takes on more people.

2.5.1.1 The Endogenous View

It is instructive to consider an important aspect of system dynamics modeling—the endogenous point of view. This perspective says that all components necessary to generate problem behavior are contained within the system boundary. Hence, the focus is internal, not looking for external blame. The contrasting internal and external views are frequently a source of controversy to be resolved. The external view is often event oriented, considering the system to be at the mercy of outside forces. Rather, events should be seen in the context of patterns, and structural reasons for the patterns searched for. System behavior is the consequence of its structure.

The internal view is essentially an axiom. Internally looking for behavior patterns from system structure is thought to have much more explanatory power than the external view. The focus of modeling is inward once the boundary is drawn. The existence of feedback loops within the boundary is essential. Without them, causal links would be externally connected and the causes of behavior would be traced to outside forces. This is not to deny that external influences can have important impact on a system. Sometimes, these external forces may be a focus of study, but in that case the system boundary has been redefined to include them.

2.5.2 Causal Loop Diagrams

Causal loop diagrams (also called "causal diagrams") are simple diagrams with words and arrows that help portray cause and effect relationships and information feedback in a system. A loop refers to a closed chain of cause and effect. Used carefully, these diagrams can be very effective in explaining how dynamic behavior patterns are generated. The usage of causal loop diagrams is sometimes debated, but ultimately their effectiveness depends on the context. They may be very useful or they may be practically superfluous, depending on the goals, constraints, and participants of a study. They are generally valuable during the top-level formulation of a fuzzy system concept, but elaborated stock and flow models show additional concepts and are more precise. The primary drawback is that causal diagrams obscure the stock and flow structure of systems and lose the crucial concept of accumulation.

Causal diagrams can be misleading due to the lack of accumulations and flows. Causal diagrams make no distinction between information links and rate-to-level links (conserved flows). When there are rate-to-level links, this simplification causes false characterizations of positive and negative polarities in causal loops. Even experienced modelers can easily come up with different (and erroneous) behaviors from the same causal diagram. Using a stock and flow structure will increase the number of correct interpretations of system behavior. Caution should be used since causal diagrams can be easily abused.

In early system dynamics, feedback structure was portrayed with equations and stock and flow diagrams. These representations are natural for engineers, but causal loop diagrams were created to serve a wider range of people and have become popular since then. Causal diagrams support free-form, creative thinking, whereas stock and flow diagrams are more explicit and support analytical thinking. Causal diagrams are often appropriate to begin group modeling exercises. Others use causal diagrams after the fact to illustrate the high-level concepts of an elaborated stock and flow model.

With modern visual programming utilities for system dynamics models, experience has shown that causal diagrams have limited staying power when elaborated, executable models are desired. They quickly lose relevance when more detailed concepts and equations start being fleshed out since there is no distinction between levels, rates, auxiliaries, and so on. Elaborated stock and flow models show more than what causal diagrams can, so they are more powerful to communicate with if the audience understands them. Frequently, the model elaboration process shows that the initial causal diagrams were too simplistic or incomplete to explain a system and its behavior. In many situations, causal diagrams can be bypassed and an elaborated stock and flow model can be started immediately.

On the other hand, some studies will entail only causal diagrams, with no need for further model elaboration. If they satisfy the goals of a study then it is certainly valid to stop there. Despite their drawbacks, people at all levels will probably continue to rely on causal diagrams to communicate feedback structure. Their most appropriate role appears to be for public consumption. A classic article on causal diagram issues is George Richardson's "Problems with Causal-loop Diagrams" [Richardson 1986].

2.5.2.1 Diagramming Conventions

There are several similar conventions for drawing causal loop diagrams. They all use words and arrows to identify system entities and causal connections. Symbols denote the polarity of the connections and the types of feedback loops created (i.e., positive or negative loops). The following outline for creating causal loop diagrams describes a basic method with minor variants. This method is illustrated in the following steps that lead up to an enhanced causal diagram for Brooks's Law.

Step 1. Use words to represent the variables in the system that you wish to interrelate. Use nouns and avoid using verbs or action phrases that indicate a direction of change, such as "increasing" or "decrease in . . ." Arrows will convey the actions and their signs will indicate whether variables increase or decrease relative to each other. Figure 2.15 shows a couple of simple causal relations.

Try to use the more positive sense of a variable name so that relationships are clearer. Take, for example, the aspect of increasing project overhead due to added person-

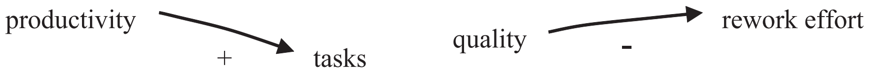


Figure 2.15. Simple causal relations.

nel per Brooks's Law. The term "project overhead" is preferable to "increasing project overhead" because a decrease in "increasing project overhead" is confusing (see Figure 2.16). Or if "motivation" goes up or down, it is easier to convey an increase or decrease in "lack of motivation."

For every action included in the diagram, try to think of unintended consequences as well as expected outcomes. For example, "schedule pressure" may increase productive time on the job but also induce more errors. It does not matter at this point whether the variables are levels, flows, or auxiliaries. But they should represent quantities that can change over time.

Distinguish between perceived and actual states, such as "perceived quality" versus "actual quality." Perceptions often change slower than reality does, and mistaking the perceived status for current reality can be misleading and create undesirable results. For example, determining if a software product is ready to release may be erroneous if defect reports are not being written or there are substantial reporting delays. The decision maker assumes that there are few defects when the reality is that there are many defects that he is not aware of.

If a variable has multiple consequences, aggregate them first by lumping them into one term while completing the rest of the loop. For example, we started illustrating an aspect of Brooks's Law in Figure 2.16 by showing that added people cause a change in "project overhead." Next we can add the causal effect of project overhead on productivity by adding a positive sense arrow from project overhead to productivity.

In order to represent the different aspects of project overhead, we split it into communication overhead and training overhead. Figure 2.17 shows this disaggregation with both of the effects on productivity.

Step 2. Arrows stand for causal connections between the variables. Draw arrows that point from causes to effects, completing loops wherever possible. In some conventions, a "//" is used to interrupt any arrow that represents a process that contains a significant delay. Some methods distinguish between long-term and short-term consequences of actions. Larger loops are used as the consequences progress from short-term to long-term processes.

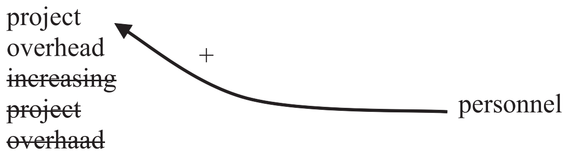


Figure 2.16. Causal relation using positive sense for personnel project overhead.

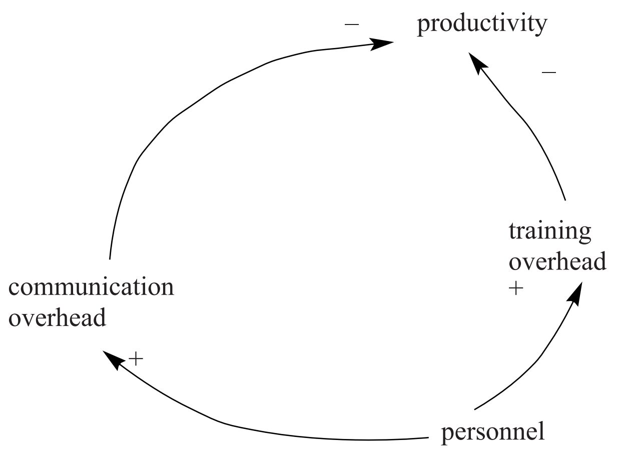


Figure 2.17. Disaggregating project overhead causal effects on productivity.

Step 3. Label the connection polarities at the arrow tips. To determine the polarity for each arrow, either increase or decrease the cause (i.e., the variable at the tail of an arrow). Then, determine whether the effect (i.e., the variable to which the arrow is pointing) moves in the same or opposite direction. If the effect moves in the same direction, use a "+" sign to indicate a positive polarity. If the effect moves in the opposite direction, adorn the arrowhead with a "–" sign.

Note that some diagramming conventions use different pairs of polarity signals. One convention uses the letter "s" to indicate "same" and an "o" to denote an "opposite" effect of one variable relative to another. Sometimes, a positive connection has a blank label and a negative one denoted by a U-turn symbol. The advantage of the latter approach is that the diagrams are less daunting with one-half as many polarity signals. The U-turn symbol is also more recognizable for the inverting concept.

If a link between two terms requires a lot of explanation to be clear, redefine the variables or insert an intermediate term. For example, the relationship between "market window" and "defects" may be more obvious when "schedule pressure" is inserted between them. A decreasing market window increases schedule pressure, which causes more defects (and thus lower quality).

Step 4. Trace the direction of change around each loop in the diagram, beginning with any variable in it. If, after cycling around the loop, the direction of change of the starting point variable is in the same direction as the initial direction of change for this variable, place a (+) in the loop center to indicate a positive (or reinforcing) feedback loop. If, after propagating around the loop, the direction of change of the starting point variable is opposite to the initial direction of change, place a (–) in the center to indicate a negative (or balancing) loop.

There is a shortcut for determining whether a loop is positive or negative. Count the number of negative signs (or "o"s) in the loop. An odd number of negative signs indicates a negative or balancing loop (i.e., an odd number of U-turns keeps you headed in the opposite direction). An even number of negative signs means it is a positive, reinforcing loop. After labeling the loop, you should always read through it again to make sure the story agrees with the loop type label.

Negative feedback loops are goal-seeking processes (hence they are also called balancing loops). Try to make explicit the goals driving the loop. In the Brooks's Law situation, by identifying "planned software" as the goal, we see that the "schedule gap" (which is reflected in the delta between planned and actual developed software) is really driving actions to add people. Figure 2.18 is an enhanced causal loop diagram of Brooks's Law effects with a goal-seeking feedback loop to close the schedule gap.

There are many more elaborate treatments of causal loop diagramming in the system dynamics literature. Hart describes a method in [Hart 2004] with different software process examples.

2.6 MODEL FORMULATION AND CONSTRUCTION

Model formulation elaborates the conceptual system structure into detailed equations for simulation. The system of interest is described in a model consisting of levels, rates, auxiliary variables, and feedback connections. This transformation to a more precise and quantitative state forces a clarity of thinking that continuously improves understanding of the system under study. Formulation and construction of a model involves iterative elaboration, construction, and simulation activities. This section will describe detailed model development in which the pieces are put together, whereas Section 2.7 focuses on executing the progressively complex constructed models.

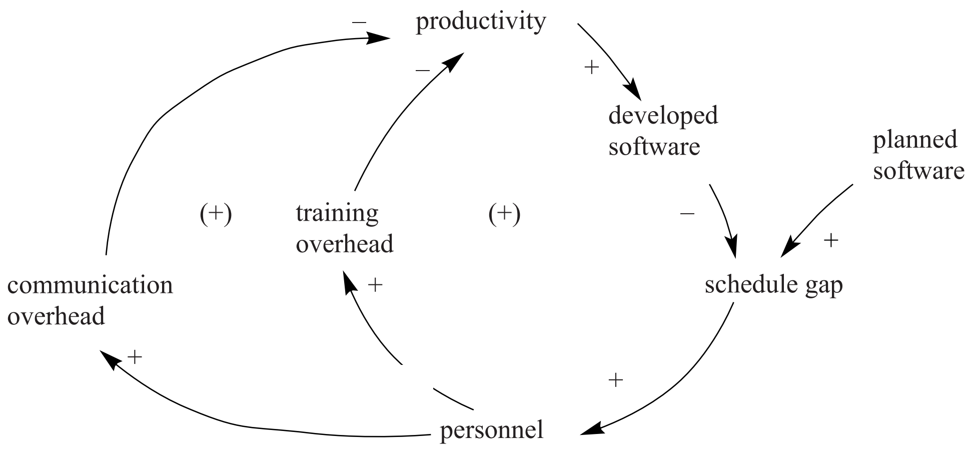


Figure 2.18. Brooks's Law causal loop diagram with goal-seeking feedback.

Just as a group of minds is used to conceptualize a model in the beginning, other participants are also needed during model formulation to describe the current or desired processes. This continuation of the stakeholder communications started during problem definition and model conceptualization often involves people in different roles than those first engaged. Conduct interviews and meetings with those executing the processes. An important heuristic to keep in mind is, *Don't model in isolation; try to involve those being modeled.*

2.6.1 Top-Level Formulation

The top-level formulation activities continue model conceptualization and overlap it to some extent. Conceptualization and top-level model formulation are the iterative elaboration activities before detailed model construction and assessment. Top-level structures serve as an intermediate design bridge between conceptualization and detailed construction. In system dynamics, this intermediate artifact is often represented in an executable model with appropriate stubs and initial approximations.

A high-level mapping of the system is the desired result in this stage. Things should be kept relatively simple. The formulation process can be started by looking for a main chain in the system, which is a sequence of stocks connected by conserved flows. As systems are composed of flows and their accumulations, ask the question, What is accumulating in the system?

2.6.1.1 Identify Major Flow Chains and Model Sectors

Identify the single most important accumulation (i.e., stock or level) in the system. Is it software artifacts? Is it people, defects, effort, or something else? Then define the flows associated with the accumulation(s). Look at each flow in isolation. Ask, What is the nature of the activity generating the flow?

Tasks and people accumulate in the Brooks's Law situation. Since tasks and people are quite distinct entities, they should be modeled as separate flow chains. The two flow chains constitute the backbone of the Brook's Law model upon which specific rate equations and auxiliaries are built on top of.

If a main chain is not apparent, one method that can help is to focus on the key actors in the system. Identify a small set of actors (not always human) that constitute a class of individual elements. These are high-level model sectors. Sectors often correspond to functional areas within a business enterprise, and the key actors are the process stakeholders in those areas. For each actor, try to assess the following:

- What conditions does this actor monitor?
- What actions are taken when conditions need to be adjusted?
- What resources are used when the actions are taken?

The conditions may eventually be elaborated into levels, actions will become flows in the system, and resources will also translate into levels.

Sometimes, pictures can be used to illustrate important model sectors or key actors. These pictures will help in the overall model understanding. It is fruitful to see how various stakeholders view the system differently. The visualizations can be important to help understand and reconcile differences between stakeholders.

Examples of model sectors for the software process include software development, quality assurance, and planning and control. The software development sector works off of project goals and constraints. Planning and control monitors the output of software development, and may revise project goals and constraints (e.g., new deadlines) and assign more resources. Quality assurance monitors the products of software development and makes adjustments by spending more or less time on artifact review. All sectors use personnel labor resources.

Generally, you do not want to include each and every key actor or sector in the initial formulation. Typically, the sectors represent aggregations of smaller elements. As you continue to elaborate the model formulation, you see how far the initial aggregations and characterizations take you.

2.6.1.2 Layout Plumbing and Generic Flows

Try to "lay out the plumbing" of the system in terms of putting down stocks and flows without extraneous connectors. You lay out the core infrastructure upon which further details are added later. After each stock is deposited into the system, ask what are its respective inflows and outflows. When possible, use generic infrastructures like those discussed in Chapter 3. Using existing structures that fit the system is cost efficient reuse. The core structures will be used to define relationships that are thought to be causing the dynamic phenomena under study.

The basic plumbing connections are then elaborated into more detail. The flows should be characterized and information loops should be closed. For each flow in the system consider the following:

1. What is the nature of the process?
2. How does the process really work?
3. How are decisions made?

Do not focus on "what are the inputs to the process"; instead consider how you would actually make a decision yourself.

Generic flow processes (provided in Chapter 3) should be considered for each flow. It is very rare to have a structural element that has not been modeled before. When using a generic flow process, try to not violate its integrity. Other inputs can be brought into the structure of an existing generic process. Ensure that the units of each flow match its associated levels divided by time. Use real-world variables instead of creating conversion factors for the equations to match.

The last step of fleshing out the plumbing is to close loops. Examine any converters that do not have connections to them. Determine whether they should be in a feedback loop or not. Make sure that outflows from a level are linked to that level with a connector chain that does not pass through other levels. Such a connection could allow a level to go negative.

Model elements consisting of levels (stocks), rates (flows), sources and sinks, auxiliaries, and feedback connections were introduced in Chapter 1. The following subsections provide more detail to interpret and apply these elements during model formulation.

2.6.1.3 Levels

A level is an accumulation over time, also called a stock or state variable. It can serve as a storage device for material, energy, or information. Just as the instantaneous level of water in a storage tank measures the time accumulation of its inflow and outflow rates, the level of software waiting to be tested is the accumulation of developed software minus that already incorporated into system testing.

Levels exist in conservative subsystems since they store conserved quantities of flowing entities. They can be changed in quantity only by moving the contents between levels, or to a sink or from a source. Contents move through levels via inflows and outflows. A level can have multiple inflows and outflows. These flows are the *rates* described in the next major section. All levels must, therefore, have rates associated with them.

Levels represent the states since they are the only parameters needed to describe a system at a point in time. They are a function of past accumulation of rates, and exist at system rest. They must be initialized with a value.

A level can easily transform the pattern of behavior coming into it. Suppose the inflow to a level is constant. With no outflows, the level will show a rising pattern in a straight line. The slope of the line equals the constant inflow rate. If the inflow is rising in a straight-line pattern, the level with no outflows curves up like a parabola. Both of these are seen in the general behaviors in Table 2-2. Oscillating inputs to levels cause similar oscillating outputs that are shifted in time.

In contrast to a level, if the inputs to an auxiliary equation are constant then the auxiliary is too. If the auxiliary inputs oscillate, so will the auxiliary itself but it will not be shifted in time like a level. There are no delays when quantities change for auxiliaries (or rates).

There are some modeling implications for tightly linked variables. If variables in a causal loop are tightly linked in their behavior, they may be modeled as auxiliaries. If one of the variables can exhibit a different dynamic behavior, then a level variable is indicated somewhere in the chain. It will decouple the ends of the auxiliary chain and allow them independent behavior.

A level also serves to decouple rates. Consider the purpose of inventory: it is a buffer, or decoupler, between production and orders so that if one of them fluctuates the other can stay relatively smooth. By decoupling production and shipment rates, they can vary independently over time.

All level variables with inflows and outflows have the capability to decouple rates. By rising and declining when necessary, the level absorbs the differences between the inflows and outflows.

2.6.1.3.1 Multiple Levels and Oscillation. Levels can also cause states of disequilibria. Previously, it was noted that levels can be fluid couplers between causal chains or decouplers of inflow and outflow rates. Because of these characteristics, adding a level can cause disequilibrium in the system. Equilibrium is when inflows are balanced with outflows in a dynamic system. Since levels can decouple, then adding a level makes it possible for inequalities to exist.

Suppose a model has a single level. It stops changing when its inflows and outflows are equal. With no other levels, all the variables in the model are traceable to the level and they remain constant from the point at which the flows are made equal. If another level is added, then both sets of inflows and outflows must be balanced for total system equilibrium. Otherwise, there could be oscillations as the second level perturbs the equilibrium.

Thus, a model with a single level cannot oscillate, but a model with two or more levels can. This is because a second level provides an opportunity for one part of the system to be in disequilibrium. For example, suppose there are separate personnel levels to represent different projects and proposal activities. When new projects come on board and hot proposals request people from projects, there is a distinct possibility of volatile staff levels continuously rising and falling to address the perceived needs. New project staffing needs are not balanced in conjunction with other activities—a policy of poor communication. Chapter 3 shows applied examples of oscillating systems.

2.6.1.4 Time Horizon Considerations

There is a conceptual connection between the model time horizon and the time for which different variables actually change. Something may be well represented as constant when it is essentially fixed for the simulation time horizon, but it may change during a longer time frame. Quantities are modeled as constant when they do not change for the applicable time horizon, not because it never changes. Some constants represent accumulations that are changing too slowly to be significant. Conversely, an accumulation that is very fast for the time horizon can be modeled as an auxiliary (if saving computations is important).

It has been stated that all levels represent accumulations, but the converse is not always true. Some concepts may be better suited as constants or auxiliaries rather than levels. The "snapshot test" described next may provide guidance for deciding what to model as a level.

2.6.1.4.1 Snapshot Test. Pretend to take a snapshot of the actual system under study. Stop time and freeze flows in the system as if you took a photograph. Again, it is useful to consider a water analogy of a running stream and intermittent pools. One could measure the height of pools in a static photograph but not the rate of water flow between them. Level variables are those that still exist and have meaning in the snapshot; the accumulations can be measured.

If time were frozen on a software process, production rates could not be measured but the number of produced artifacts could be determined. This signifies that the number of developed artifacts is suitable to be modeled as a level, with the artifact production rate being an inflow to it.

Suppose a snapshot is taken during the middle of software construction. Software coding rates would cease, but a measure like average productivity would continue to exist and could be determined. With units of function points/person-month, the variable seems more like a rate. But averaging involves accumulation over time. Thus, units are not always helpful for selecting level variables. The snapshot test may be difficult to apply in averaging or perceiving situations. The time constant associated with the accumulation is also considered, to ensure that the accumulation is not changing too fast or too slow to be modeled with something other than a level variable.

Not all quantities that continue to exist in a snapshot should be modeled as levels. Suppose one has a constant workforce that will not change during the time horizon. Even though the workforce was put into place and accumulated through hiring and attrition, it can safely be modeled as a constant if the determination is made that it effectively will not vary for the purposes of the study. One example is a very short project with a fixed staff size in an organization with very low attrition.

2.6.1.4.1.1 Example: Snapshot Test. Suppose we take a snapshot of a running project for the Brooks's Law model. The following elements can be measured and are eligible to be levels (accumulations):

- The amount of software still to be developed
- The amount of software already developed
- The number of people working on the job, both new and experienced

It is important to account for different experience levels, so both new and experienced personnel warrant separate levels. If they are combined into one, then important effects for learning and training cannot be handled.

Model elements that cannot be measured at an instant of time and are best modeled as rates are:

- The current productivity rate, or "project velocity"
- The rate of new people coming on board the project
- The rate of people becoming experienced

2.6.1.5 Sources and Sinks

Sources and sinks indicate that flows come from or go to somewhere outside of the model. Their presence signifies that real-world accumulations occur outside the boundary of the modeled system. They represent infinite supplies or repositories that are not specified in the model. The concept of a system boundary is crucial to all system modeling—it must be unambiguously defined and it drives modeling decisions. Sources and sinks must, therefore, be carefully considered and revisited throughout the modeling process. The implications of replacing them with levels should be given careful thought. See Section 2.5.1 for guidance on identifying system boundaries.

2.6.1.6 Rates

Rates are also called flows—the "actions" in a system. They effect the changes in levels and are thus inseparable from levels. They must occur together. Rates are computed as a function of levels, constants, and auxiliaries. Rates may be bidirectional. Levels and rates are both necessary to model a system. Rates only interact through their influence on system levels.

Rates represent decisions (for action and change) or policy statements. They are based only on available or apparent information. This information may be delayed in the real world, since levels changes often occur before information on them is available for decision making. You must distinguish between desired and actual conditions, as well as between actual and perceived conditions, since decisions are made relative to perceived conditions. Thus, rate equations for making decisions must account for perception delays and other biases that exist in the real system. All rate equations should make sense even under extreme or unlikely conditions.

To a large degree, the essence of system dynamics modeling is defining levels and rates, and then developing the rate equations. It is from the rates and initial values of levels that everything else is calculated. In some studies, it may become difficult to differentiate level and rate variables in systems. The snapshot test previously described is often useful for identifying rates versus levels.

2.6.1.7 Auxiliaries

An auxiliary variable signifies information in the system. As model elements, they make relevant information accessible. Though it may be possible to develop a system dynamics model without auxiliaries, since all information is traceable to levels and rates could be formulated with constants and rates, the resulting model would be unreadable and hard to follow. Important system information is captured in auxiliaries that help in the formulation of rate equations. They can take on a wide range of meanings.

Auxiliaries are converters of input to output, and help elaborate the detail of stock and flow structures. An auxiliary variable must lie in an information link that connects a level to a rate. Auxiliaries often represent "score-keeping" variables, such as the percent of job completion.

Auxiliary equations represent algebraic computations like rates, and have no standard form. Depending on the functional relationship, an auxiliary can be computed from constants, levels, rates, or other auxiliaries.

When translating a system concept into quantitative terms, deciding on levels and their affecting rates is a first step. The remaining concepts are typically formulated as auxiliaries. To do this, identify the units of the concept. Understand what it means if it rises or falls. Next, identify the units of the variables that will determine the concept. Sometimes, looking ahead in the model will help identify the units. It is important to note the distinction between a quantity and its effects in the system. The variable should be formulated before trying to quantify its effects.

If an auxiliary is a function of one or more quantities and their units match, then an additive or subtractive formula is a good bet. If the units of an auxiliary are different than those of the variable that determine it, then multiplication or division is likely. Examples of these different auxiliary formulations are shown in Chapter 3.

2.6.1.8 Connectors and Feedback Loops

Information linkages are used to represent information flow (as opposed to material flow). Rates, as control mechanisms, often require connectors from other variables (usually levels or auxiliaries) for decision making. For example, knowledge of the progress of tasks tested may influence a decision represented in a controlling rate (e.g., to move resources from a less critical path to testing). Information links are the only inputs to rate equations.

A feedback loop is a closed path connecting an action decision that affects a level and information on the level being returned to the decision-making point to act on. Such a loop exists if a rate connection represents information based on previous decisions. The elements of a feedback loop are a decision (or action or rate), an affected level, and an information path connecting the level to the action. See Figure 2.19 for an example.

The example in Figure 2.19 is the simplest form of feedback without additional delays, information distortions, or interconnected feedback loops. It is important to realize that only available, *apparent* information is used for a decision basis. Feedback data may be late or erroneous. Such imperfections can lead to poor decisions and project downfalls, as will be detailed in later sections.

Feedback mechanisms whereby decisions made at one point in the process impact others in complex or indirect ways must be accounted for. In a software development process, the decision to hire a new employee has multiple impacts and implications over the entire course of the project. The decision to inspect requirements or not also has many implications. For complex feedback or feedforward systems, an analytic solution may not be feasible, so simulation enables a more useful alternative.

2.6.2 Basic Patterns and Rate Equations

Formulating rate equations is a primary focus of model formulation. All system variables (except constants) are ultimately calculated from rates integrated over time in the simulation tools. Rate equations translate system pressures such as planning policy into actions that alter the system state. Typical rate formulations, or patterns, are described in this section.

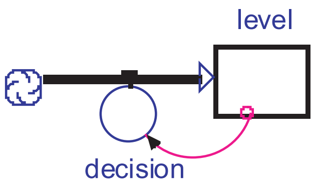


Figure 2.19. Feedback loop.

2.6.2.1 Constant Rate with a Level

The basic rate equation below multiplies a constant and a level:

$$\text{rate} = \text{constant} \cdot \text{level}$$

The constant may take on several interpretations that impact the resulting dynamic system behavior. This equation is suitable for modeling a simple production function as follows:

$$\text{software development rate} = \text{productivity} \cdot \text{personnel}$$

An example is the production structure shown in Figure 2.20 that connects a task and personnel chain.

This is one of the basic infrastructures presented in Chapter 3 and is the core of the Brooks's Law model software development rate formulation. Progressing beyond the assumption of static productivity, the constant is replaced with a variable function such as in the Brooks's Law model. The modeling goals and constraints will dictate the level of detail in formulating the rate.

2.6.2.2 Variable Rate with a Level

An extension of the constant rate equation is to replace the constant with an auxiliary variable:

$$\text{rate} = \text{auxiliary} \cdot \text{level}$$

This structure will model more realistic situations, such as when productivity is not constant over time. The initial Brooks's model is a good example. An auxiliary variable is needed to adjust the overall productivity for communication overhead, training, and learning. Even without complicating project factors like new hires, a learning curve applies in most situations. Learning curves are addressed further in Chapters 3 and 4.

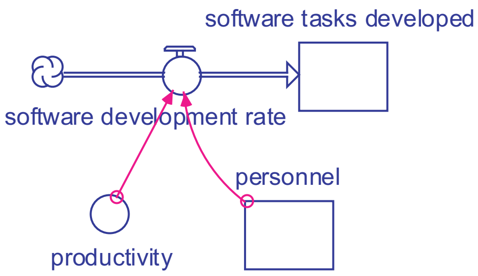


Figure 2.20. Example software production structure.

2.6.2.3 Positive (Reinforcing) Feedback Growth or Decline

Positive feedback is reinforcing feedback that tends to amplify movement in a given direction. Positive feedback often produces a growth or decline process, such as population growth. A simple modification of the linear rate equation uses feedback from a level to its inflow rate to produce positive exponential growth or decline. Positive feedback can be formulated with the following equation:

$$\text{rate} = \text{growth fraction} \cdot \text{level}$$

The structures shown in Figure 2.21 produce exponential growth behavior.

This structure produces exponential growth over time, as seen previously in Table 2-2, which shows general model behaviors. Compared to a standard linear rate equation, the constant now represents a growth fraction and its value determines the dynamic behavior. This formulation fits when the real-world process exhibits a rate that increases in proportion to its level.

Using population growth as an example, the birth rate depends on the current population. The number of births increases as population increases. A birth rate increase leads to more population. If population declines for some reason, the opposite occurs. The up or down direction is reinforced. The term positive feedback can sometimes be confusing since it can produce a negative trend, so reinforcing feedback is often used instead. There are analogous positive feedback growth situations in the software industry. See Chapter 3 for specific software process examples.

Positive feedback is analytically formulated per the following formulas. Exponential growth is represented as level = $\text{level}_0 e^{at}$. Exponential decay is written as level = $\text{level}_0 e^{-t/TC}$. The doubling time for exponential growth is $0.69/a$ and the half life for exponential decay is $0.69 \cdot TC$, where TC is the time constant.

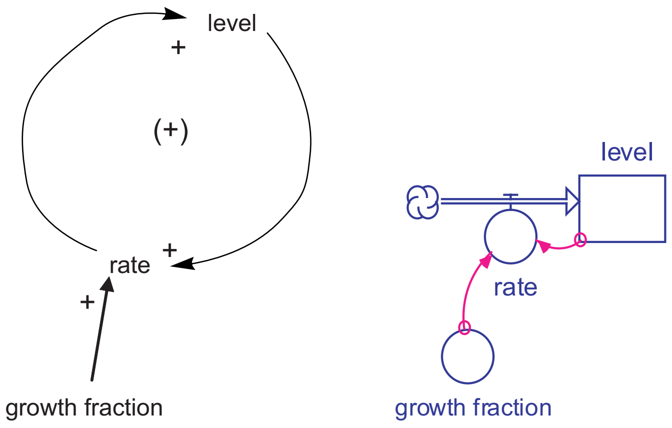


Figure 2.21. Exponential growth structures.

2.6.2.4 Delays

Time delays are ubiquitous in processes, and are important structural components of feedback systems. Examples include delays associated with any complex activity (or life-cycle process step) performed by resource-limited teams, hiring delays, problem resolutions, change approvals, and so on.

The world has a lot of situations in which a delayed outflow from a level can be represented by

$$\text{rate} = \text{level}/\text{delay time}$$

Figure 2.22 shows the delay structure. The delay time is the average lifetime an entity stays at the level. It may signify the average time an order is in backlog before being shipped, or the average time that a software module remains in the design stage. The delay constant thus has real-world meaning. Another advantage of this formulation is that an outflow rate cannot cause its level to go negative, which is another appropriate model of reality.

As an example, the formulation for new hire assimilation in the Brooks's Law model uses this simple delay structure. The average time for assimilation (20 days) becomes the delay time, so that the number of people transitioning to experienced in a time interval is 1/20 of the current level.

The delay time constant may be replaced with an auxiliary variable, just as the first rate equation in Section 2.6.2.1 used a constant and was elaborated into a variable in Section 2.6.2.2. Thus, the variable delay equation takes on the form

$$\text{rate} = \text{level}/\text{auxiliary}$$

where the auxiliary now represents variable delays, lifetimes, and so on.

Delays are parent objects for a number of other structures including decays, exponential decays, and cascaded (compounded) delays. They are common components of other substructures, including goal-seeking feedback. See Chapter 3 for more discus-

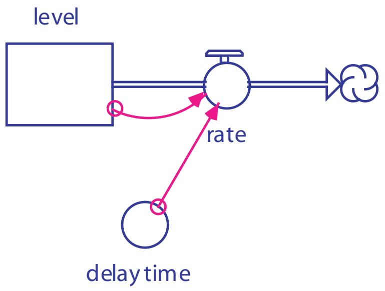


Figure 2.22. Delay structure.

sion of delays, infrastructures that contain them, and real-world interpretations of the delay time constant.

2.6.2.5 Negative (Balancing) Feedback

Negative feedback is probably the most classic feedback loop in human systems. It is a structure that strives to bring the system state closer to a desired goal:

$$\text{rate} = (\text{goal} - \text{present level})/\text{adjustment time}$$

The adjustment time is a time constant representing the period over which the rate tries to close the gap. Negative feedback exhibits goal-seeking behavior in which the change is more rapid at first and slows down as the discrepancy between desired and perceived decreases. Goal-seeking negative feedback systems abound, so the above structure is a powerful general pattern for many rate equations. Figure 2.23 shows a negative feedback structure with both a causal loop diagram and a level/rate model. The analytic expression for this negative feedback is

$$\text{level} = \text{goal} + (\text{level}_0 - \text{goal})e^{-t/TC}$$

where level_0 is the initial level value and TC is the time constant or adjustment time. Negative feedback is balancing feedback that tends to offset or balance movement in a given direction, often described as a goal-seeking loop or control process. Common examples are heating systems or other homeostatic devices (such as the governor on a bus or lawnmower). When balancing feedback is present, the signal or information that is fed back is opposite in sign, such that it tends to cause the system to resist the change. Sometimes, the term negative feedback can be misleading, so balancing feedback is more precise. Chapter 3 has software process examples.

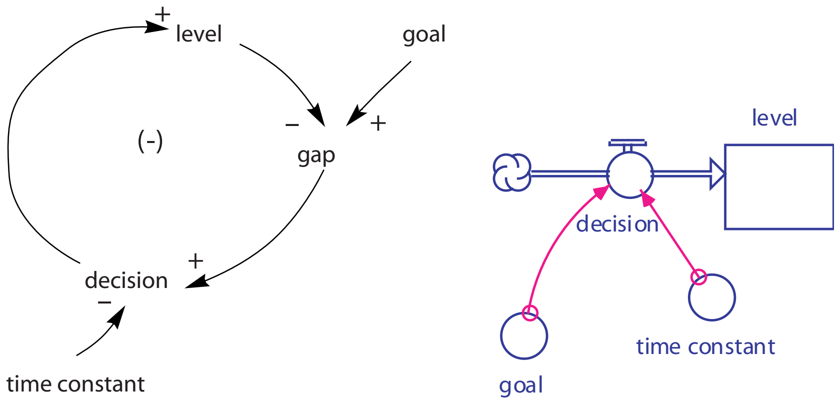


Figure 2.23. Negative feedback structures.

The previous rate pattern, rate = level/delay, is a special case of the net rate pattern for negative feedback. The difference is that the "goal" in the delayed outflow example is zero level. Table 2-2 shows behavior for negative feedback with a positive goal. For a zero goal, the graph is flipped upside down as the level starts high and approaches zero.

2.6.2.6 Addition and Multiplication of Effects

The rate patterns described above are the building blocks for more detailed equations. Two more complications arise when a rate is adjusted by the addition of more effects or when it is multiplied. The addition rate is

$$\text{rate} = \text{normal rate} + \text{additional rate(s)}$$

The terms added together must represent well-understood process concepts that are truly additive. A simple case is a personnel hiring rate accounting for both the natural quit rate and the rate of adjusting the workforce to a desired level to meet project demands:

$$\text{hiring rate} = \text{average quit rate} + (\text{desired staff} - \text{staff})/\text{hiring delay}$$

A multiplicative effect is modeled with

$$\text{rate} = \text{normal rate} \cdot \text{multiplier}$$

This structure is valuable for modeling a good number of situations in which a normal flow rate is multiplied by one or more factors. The multiplier is easily conceptualized, understood, and handy if it represents a percentage change. For example, effort multipliers are well known in other modeling situations like static cost models. Linear factors are used that use unity as a nominal or reference value.

A value of one for the multiplier represents the normal situation. A value of 0.8 for example signifies a 20% reduction in the rate, and 1.5 represents 50% additional flow. The basic formula can be made more complicated with additional factors. When such factors are truly multiplicative, a cascade of factors can be used to represent the situation:

$$\text{rate} = \text{normal rate} \cdot \text{multiplier } 1 \cdot \text{multiplier } 2 \cdot \ldots \text{multiplier } n$$

The Brooks's Law model has this type of formulation for the software development rate. The rate is adjusted multiplicatively with factors for communication overhead, weights for personnel mix, and an adjustment for the effective number of personnel.

When formulating additive or multiplicative rate equations, some principles should be kept in mind. Multiplication can shut off a flow rate completely if a multiplier takes on a zero value. Addition will not zero out a rate except when the added terms are negatives of each other. Addition can be thought of as cooperating or contributing to a rate determination. Multiplication can, however, be dominating in its influence. If one is trying to decide between the two, consider whether the different combining influences cooperate or whether one dominates the other. If an effect can dominate, then it should be modeled with multiplication.

2.6.2.7 Coincident Flows (Coflows)

Coincident flows (coflows) occur simultaneously (in parallel) through a type of slave relationship. An information connection is used to model one flow driving another. A general equation for a coflow is

$$\text{rate 2} = \text{rate 1} \cdot \text{conversion coefficient}$$

A prime example in the software process is defects being generated in unison with software development. A convenient way of expressing the defect generation rate is multiplying the defect density (the number of defects normalized by size) by the production rate, such as in the example for software design in Figure 2.24. The design defect generation rate is slaved to the design rate per the design defect density.

The equation for this coflow is

$$\text{design error generation rate} = \text{design rate} \cdot \text{design error density}$$

2.6.3 Graph and Table Functions

A graph or table function is a simple and convenient way to specify variable relationships. The graphical version is easy to understand through its visualization. The slope, shape, anchor points, and reference lines in a graph require careful consideration. Some guidelines for formulating graphical functions or their table equivalents are discussed below.

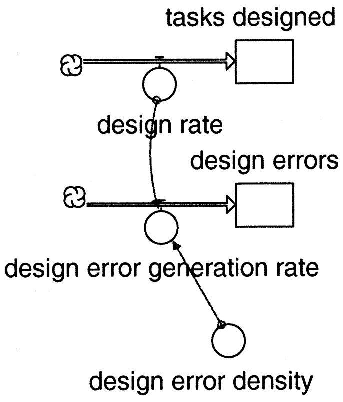


Figure 2.24. Example defect generation coflow.

Set the slope of the relationship to the polarity of the effect it represents. A negative effect should inspire a negative slope, and a positive slope denotes a positive relationship between variables. The shape of the curve should be evaluated in terms of the slope and curvature at the middle and extremes. The flattening of a function corresponds to a weakening or saturating effect. An example of this is the tail of an S-curve representing a saturation point, like a saturated market. Steepening a curve represents the strengthening of an effect.

It is highly recommended to normalize or make the graph relative by formulating it as a function of the ratio of the input to its reference value. The nominal reference point is then clearly indicated where it equals one.

Sometimes, when quantifying information in feedback loops, the modeler must mentally shift from conceptual thinking to algebraic thinking. Considering the units of the quantities involved can help lead the way. Dimensional analysis, which is the manipulation of units as if they were algebraic quantities, is a valuable skill for this.

Graph and table functions are highly useful for formulating nonlinear pressures and effects. All they require are determination of the slope, the general shape, a couple of reference lines, and points to define a function. Normalize the inputs to table or graphical functions. Try to pick meaningful lower and upper bounds for the range, then pass smooth monotonic curves through the range.

2.6.3.1 Example: Overtime Multiplier Function

This example will create a normalized graph function for an overtime effect. It is an approximation with a limited range that will be extended further in Chapter 5. Suppose that management pushes for overtime in order to catch up on schedule, and that the management edict can be expressed in terms of desired output to normal output. We wish to model an effective overtime multiplier for productivity. How should this relationship be modeled? Limits exist such that people will not continue to increase their overtime past their saturation thresholds. Thus, there is a value of desired output beyond which no further overtime will be worked. Studies have shown that a small amount of aggressive scheduling will tend to motivate people to produce more, but highly unreasonable goals will have the opposite effect.

A few people might work 80 hour weeks and even longer, but a more reasonable limit might be a 60 hour workweek for most industry segments. This puts an upper bound of 60/40 = 1.5 on the overtime factor. We will model the response of the aggregate system as a smooth curve (realizing that different individuals will have varying responses).

Recall that such functions are best done in terms of normalized variables, so the independent variable will be the desired/normal output. The relationship in Figure 2.25 was adapted from [Richardson, Pugh 1981], and shows the productivity overtime multiplier as a function of desired/normal output. This graph function is handy and easily used in a system dynamics model to relate variables. Moreover it is a relative relationship that does not need rescaling to fit different situations (though the desired output ratio range can be extended per an example in Chapter 4).

The relationship is captured in a model auxiliary variable either graphically or by specifying table values. Figure 2.26 shows a model representation for the auxiliary

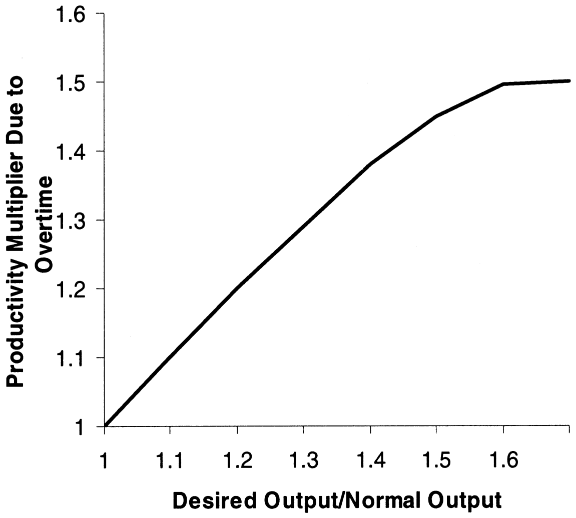
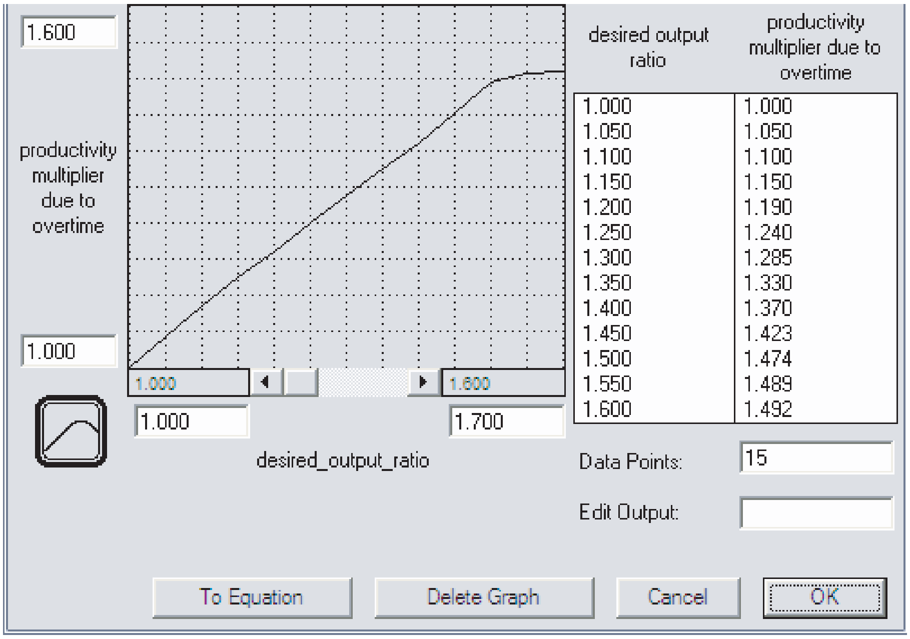


Figure 2.25. Graph function for productivity due to overtime.



desired output ratio	productivity multiplier due to overtime
1.000	1.000
1.050	1.050
1.100	1.100
1.150	1.150
1.200	1.190
1.250	1.240
1.300	1.285
1.350	1.330
1.400	1.370
1.450	1.423
1.500	1.474
1.550	1.489
1.600	1.492

Data Points: 15

Edit Output:

Figure 2.26. Model graph function for productivity multiplier due to overtime.

variable, which can be drawn and/or specified numerically. Its input is shown as the desired output ratio, but it could have two inputs for the desired and normal output. In that case the graph x-axis would be specified as the ratio of the inputs. Most subsequent illustrations of auxiliary relationships in this book will only show the graph with the table values cropped out.

2.6.4 Assigning Parameter Values

Values must be assigned to constants, graphs, table functions, and levels before a simulation can be executed. All level variables must be initialized. Rates and auxiliaries are computed from levels and constants and do not need initialization. There are several types of parameters, including constant measures, conversion factors, growth factors, delays, and adjustment times.

Every parameter and variable should have a meaningful interpretation or real-world counterpart. Parameters should have recognizable meanings to personnel familiar with the system. All parameters and variables in a model should clearly correspond to a real-world quantity (or commonly understood concept). If they are not directly observable, they should at least be intuitable to the model audience. Being careful to specify units properly will help assure that quantities are real and observable, as opposed to a parameter with contrived units. Selecting parameters is easier when they are held to standards of realism and observability.

2.6.4.1 Accuracy

The parameter values should be estimated only to the degree of accuracy called out by the study. There may be many possible purposes and ranges of accuracies required. The question to be answered is, accuracy for what purpose? Since policy analysis is generally the purpose of most studies, the parameters do not need to be estimated any more accurately than when they have no further influence on the policy. If the policy implications do not change when model parameters are varied by a certain percentage, then the parameters are precise enough for the study. Different kinds of parameters and estimates are discussed below.

Some sources for parameter estimates include:

- Firsthand process knowledge
- Quantitative relationship data that has been collected
- Quantitative data on the system behavior
- Statistical based estimates

Parameters representing measurements are usually simple to estimate because they are easily known. They can be obtained through data sources, direct observation, or knowledgeable people. One must be careful that the data used is congruent with the meaning used in the model. For example, if external defect data is used, make sure that it is reported using the same definition relevant in the model. The way quantities are used in a model will dictate how data is collected, interpreted, and estimated for the modeling.

Conversion factors translate from one unit to another (e.g., function point backfiring values convert from function points to lines of code). Normal or reference parameters require knowledge of individual relationships, such as the effect on effort of multiplicative cost drivers (e.g., product or personnel factors). These quantitative relationships may be borrowed from other types of models. As an example, the simple Brooks's Law model used multiplicative factors from static cost models to weight productivity for experience levels (in particular, the adjustment factors of 0.8 and 1.2 to account, respectively, for differing productivities of new and experienced personnel).

Some of the harder parameters to estimate include delays and adjustment times. These require cognitive thinking about the time taken to adjust perceptions, habits, and so on. Time constants are not directly observable from idealized inputs and often require mental experimentation.

2.6.4.2 Bounds

If data is lacking, the upper and lower bounds of a parameter can often be estimated. Parameter values can be estimated from detailed knowledge of a process that is modeled in aggregate, or the overall behavior of a model piece can be used. One may use aggregate data to calibrate a parameter that represents an aggregate. This involves estimating a parameter from knowledge of behavior instead of detailed knowledge of underlying processes. The bounds can also be estimated from process and behavior, and an initial trial value can be picked in the middle of the spread.

2.6.4.3 Parameters and Validity

It is always preferable (when feasible) to estimate parameters below the level of aggregation in a model, such as when using detailed knowledge of a process. When estimating from behavior though, there are potential pitfalls. One must make assumptions about equations in the model, and more assumptions means a higher probability of error. Also, estimating from behavior impacts model validity testing. This is because an ideal model will generate reference behaviors on its own, rather than using parameters made to fit the desired reference behavior. Thus, no additional confidence can be had with a good fit made this way.

Alternatively, if parameters are individually estimated from knowledge of the underlying processes and the resulting integrated model generates reference behavior, then more confidence can be placed in the model.

It is tempting to use a traditional statistical technique like correlation to estimate parameters. But one should always be wary when interpreting correlational data, as a correlation coefficient can imply the wrong polarity of a connection between parameters. This may be due to everything else not being held constant when the data is collected. The same pitfall is possible when doing multiple regression. As model behavior is more tied to its structure than parameter values, it behooves a modeler to be adept at conceptualization and formulation. These skills are more valuable than knowing sophisticated mathematical techniques for statistical estimation. Blind usage of such techniques can usurp intuition. It is also more important to have good data to begin with.

2.6.4.4 Level Initialization and Equilibrium

Levels must be initialized with beginning values. The values can be matched to historical data, numbers can be chosen to initialize the model in equilibrium, or they can be purposely set to put a model on a path of growth or decline. An important skill is knowing how to start a model in equilibrium. A good understanding of many models can be had by disturbing them from a state of equilibrium and observing the generated dynamics.

A model in a state of equilibrium means the inflows and outflows of each level match or are in balance. The levels will remain unchanged and related model variables will stay constant. Things are still flowing in the system but there is no net change to the levels. During the course of model development, selected sectors may be put into equilibrium as part of the iterative process.

Analytic procedures can be used to determine the balancing inflow and outflow values. When models become too complex, however, the procedures may become intractable. In that case, a model can be run until a state of equilibrium exists, and the appropriate flow values read from the run for subsequent value setting.

Sometimes, models should be initialized for growth. An example would be the increasing size of a system being enhanced over time or any number of Internet usage patterns. In this case, parameters should be set to start a linear or exponential growth trajectory.

2.6.5 Model Building Principles

This section will highlight and review some of the important principles of model building. Both the structure of models and the process of creating them are addressed. The principles are aids for modeling, not rigid prescriptions. Some identify pitfalls, some restate modeling philosophy, and some can even be violated under certain conditions. The experienced modeler will understand the context and be able to justify any violation of principles.

2.6.5.1 Structure

Physical quantities must be conserved in a system dynamics model. A conserved subsystem is the flow of only a single type of entity together with rates that control the flow and the levels that it accumulates in. The rates and levels must alternate in any conserved subsystem.

Conserved flows and information links are distinctly different, however. Information is not a conserved flow because it can be transmitted to other parts of the system without diminishing the source. Information to rates and auxiliary variables is not conserved through time like levels are.

Only information links can connect between conservative subsystems. The flow rates of subsystems cannot be affected directly by flows from levels in other subsystems, since material flows stay within a conserved subsystem. Rather, they can be controlled only by *information* on levels in other subsystems.

Levels can be changed only by rates. No level directly affects the value of another level. The current value of a level can be computed without the present or previous values of other levels, since a level is computed from its own past value and the flow rates affecting it over a given time interval. Every feedback loop in a model must contain at least one level. Levels completely describe the system condition at any point in time; hence, they are called *state variables.*

Rates depend only on levels and constants. Rate inputs are information links that are traceable to levels and system parameters. In some exceptions, a model may contain rate-to-rate connections where a physical rate directly determines the rate of change of a paper quantity or when the time constant of intervening levels are very small relative to other time constants. When used as information affecting other subsystems, flow rates should be averaged over a given period of time.

The units of measure must be consistent in all model equations. Any terms that are added or subtracted must, of course, have the same units. Dimensional analysis cannot prove that an equation is correct, but it can show when some equations are incorrect. Levels and rates cannot be distinguished by their measurement units alone. There are many diverse units that can be used. Within any conserved subsystem of flows, all levels have the same units of measure and all rates are measured in the same units divided by time.

Normalize measures and relationships whenever possible to make them relative. This makes things easier in the long run; the model is more scalable and easier to judge than with absolute numbers. Graphs are convenient for describing relative relationships, including normalization.

2.6.5.2 Process

Iteratively refine and slowly add relationships to a model. Do not try to enumerate all the influencing factors at first, as they quickly become unwieldy. Considering simulation model development as a type of software development, an iterative approach makes much sense. Small iterations are much easier to handle when adding model elements.

Strive for a top-down approach instead of focusing on event details. Be selective in choosing what to add to a model. Try to lump things together in classes, then slowly add new relationships. Start with aggregated patterns of behavior and add more detail when ready. Keep things simple for maximum understanding.

Do not stray too far from a simulatable model. Refrain from adding too many elements before specifying their logic and testing them. Use small incremental changes whereby only a few elements are added to the model before resimulating. A rule-of-thumb is to try to keep the number of added elements to approximately five or less. From a time perspective, do not be more than 30 minutes away from a simulatable model. Figure 2.27 from [Richmond et al. 1990] shows an empirically derived relationship between the probability of losing control of the simulation versus the number of elements added to the model before simulating.

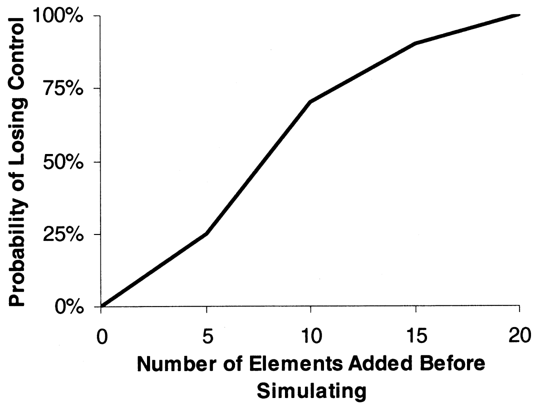


Figure 2.27. Probability of losing control of simulation.

Once you try adding about 10 elements or more, you are asking for serious trouble, and at 20 elements there is a virtual certainty of losing control. This author can personally vouch for this principle. It is like software product integration, whereby it is desired to minimize the number of changed elements being integrated together. Problems are much more tractable when an area of change can be isolated.

Before simulating, ensure that the model mechanics are finalized. Close feedback loops. Make connections between existing elements. Specify the algebra and elaborate the equations. Check the dimensional balance of equations. Ensure that all parameter values, constants, initial values, and graphical functions are set.

2.6.6 Model Integration

When integrating different sectors in an elaborate model, each sector should be tested in isolation before being combined with other sectors. As always, a steady-state situation should first be created for testing a sector. This often implies setting constants for what would normally be variable entities. There may be ghost variables (aliases) from other sectors that should be initialized to an appropriate value to simulate running the entire model. They will not change during the test simulations of isolated portions, but will vary over time when the full model is integrated.

The tests are important to make sure things are working right and to give the modeler better understanding. Yet the isolated tests are tentative in the sense that set parameters may be affected by the dynamics of the global model when it is completely put together.

Consider the two flow chains in the Brooks's Law model: tasks and personnel. The common link between the two chains is the number of personnel. The task chain can be developed in isolation if a variable for number of personnel (or two variables to denote new and experienced personnel) is used. When the two chains are working independently from each other, then they can be combined by eliminating the ghost variable(s) for personnel in the task chain and substituting the actual levels in the personnel chain.

2.6.7 Example: Construction Iterations

This example sequence of construction iterations illustrates the modeling process for the Brooks's Law model in Chapter 1. At each step, just a few elements are added and the model retested to keep its increasing elaboration under control. By adhering to the "code a little, test a little" philosophy, confidence is retained in the model because interim versions are validated before moving on. The model remains stable as it evolves. Expected behaviors are developed for each iteration and the model outputs assessed against them. Iterations of the model are shown in Figures 2.28–2.34.

Other sequences are also feasible to achieve the same end result. This approach lays down the flow chains for tasks and personnel first and then adds auxiliaries to adjust the software development rate for experience, training, and communication effect. It is also possible add some of the overhead effects to a partial personnel chain first and then complete the flow chain.

Construction Iteration #1 (Figure 2.28). Create a flow chain for software development tasks. The requirements level will be initialized, a constant value will be assigned to the software development rate, and the level of developed software starts empty. The sample project will be sized to 500 function points (the initial value of requirements) and employ 20 people for the duration. For initial testing of the software development rate, assume a mid-range value of 0.1 function points/person-day. This value is inline with published data sources and other models. The software development rate is set as such:

$$\begin{aligned}\text{software development rate} &= (0.1 \text{ function points/person-day}) \cdot (20 \text{ people})\\ &= 2 \text{ function points/day}\end{aligned}$$

With this formula, the rate will be constant throughout the simulation, the developed software will rise linearly, and the project completion time can be calculated as:

$$\text{completion time} = (500 \text{ function points})/(2 \text{ function points/day}) = 250 \text{ days}$$

The simulation is run and checked against the above calculation.

Construction Iteration #2 (Figure 2.29). Begin creating a personnel flow chain and add an auxiliary variable for nominal productivity to create a parameterized soft-

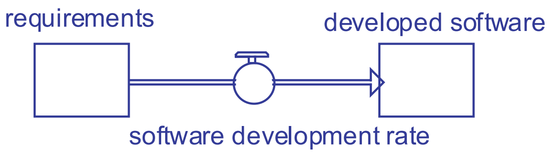


Figure 2.28. Iteration #1 structure.

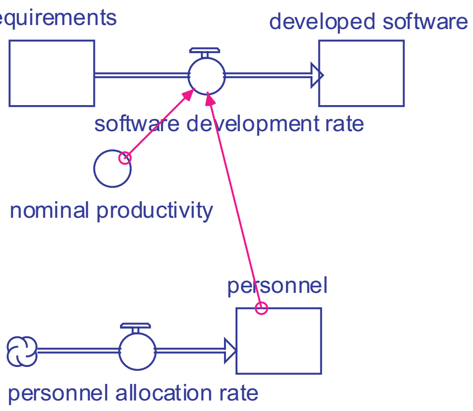


Figure 2.29. Iteration #2 structure.

ware production structure. The software development rate will depend on the number of personnel and the nominal productivity. No adjustments are made to productivity in this iteration. The personnel chain consists of a single level and inflow rate. The inflow to personnel termed “personnel allocation rate” will be used to simulate the addition of people to the project when we finally test Brooks’s Law. Other optional structures that could have been used as a placeholder for personnel, including an auxiliary variable set to a constant value or a single level with no inflows or outflows, could be initialized with a value. Either of these options would eventually have to be replaced with a stock and flow structure in a subsequent iteration.

The nominal productivity will be set to the same value of 0.1 function points/person-day and the rate equation rewritten as:

$$\text{software development rate} = \text{nominal productivity} \cdot \text{personnel}$$

Construction Iteration #3 (Figure 2.30). Complete the personnel flow chain. Another level is added to differentiate experienced personnel from new personnel. An intermediary flow between the levels is added to represent the assimilation of new people into the pool of experienced people. In this iteration, the personnel chain will be set to steady state so that there are no flows. All people will reside in the experienced personnel pool, and there will be no new project personnel. The experienced personnel pool best represents the project before new people are added. Keeping everyone in a single level will also simplify testing of the new structure and ensure that the software development rate works with no differentiation for experience levels. In the next iteration, provisions will be made for the different productivities of new and experienced people.

Construction Iteration #4 (Figure 2.31). Elaborate the productivity rate to account for the effect of personnel experience. The software development rate is linked to both levels for new and experienced personnel. This capability is first tested with the per-

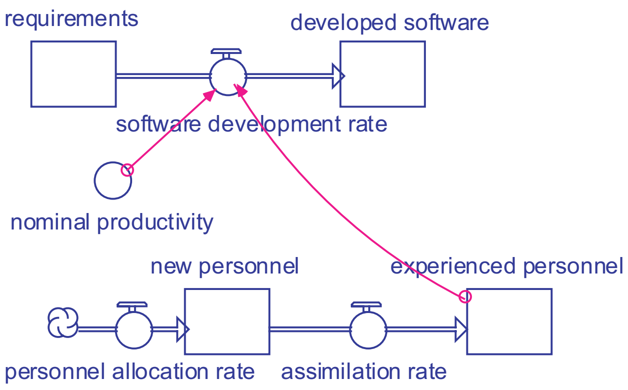


Figure 2.30. Iteration #3 structure.

sonnel chain in steady state with no flow. The portion of new to experienced personnel is varied in multiple runs to test the revised software development rate formula. A sequence of test runs could be:

1. 20 new personnel, 0 experienced personnel
2. 0 new personnel, 20 experienced personnel
3. 10 new personnel, 10 experienced personnel
4. 5 new personnel, 15 experienced personnel
5. And so on

The above values are used to initialize the levels. In steady state, they will remain at those values because the rates will be zeroed out. We will use productivity adjustment

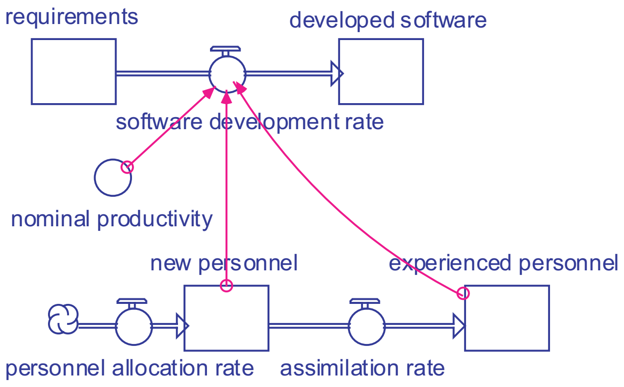


Figure 2.31. Iteration #4 structure.

factors of 0.8 and 1.2 as weights, which are in line with numerous static cost models such as COCOMO II. The equation becomes:

$$\text{software development rate} = \text{nominal productivity} \cdot (0.8 \cdot \text{new personnel} + 1.2 \cdot \text{experienced personnel})$$

The resulting productivity rate and completion time are computed independently outside the simulation and compared to simulation output for the test cases. The variables to evaluate in these runs include productivity and personnel levels (to ensure that they stay at their initial values with no personnel flow).

The productivity should remain constant throughout each simulation run because the different levels of personnel remain constant, and the accumulation of developed software representing progress will rise linearly accordingly.

Construction Iteration #5 (Figure 2.32). Make the personnel chain dynamic by allowing new personnel to assimilate into the experienced personnel pool. This will test the assimilation delay structure, ensure that the personnel chain flows are as expected, and make the productivity rate dynamic. The same set of initial values used in the test runs in iteration #4 could be used in this iteration. The same variables are tracked, paying close attention to the varying levels of personnel. A standard assimilation delay of 20 days will be used. The assimilation rate will assume this average value and be written as

$$\text{assimilation rate (persons/day)} = \text{new project personnel}/20$$

The first test run initialized with all new personnel will be the clearest indicator of whether the 20 day delay works as expected. The productivity rate will now increase over time as new personnel become more productive. The initial values of new vs. experienced will have an impact on the final completion time. In order to regression test

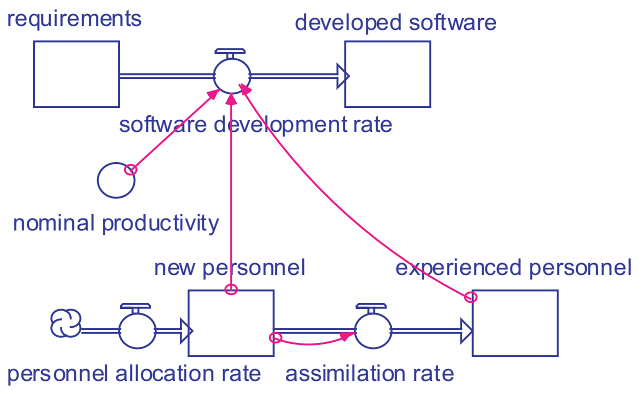


Figure 2.32. Iteration #5 structure.

and make sure previous simulation results still hold, the delay time could be made inordinately long.

Note that the addition of the delay could have been included in the previous iteration #4, but two distinct sets of iterative test cases would still have been required. The first set would revolve around testing different combinations of new and experienced (with the delay neutralized somehow), and the second set would test the delay structure.

Construction Iteration #6 (Figure 2.33). Add an effect for training of new personnel. An auxiliary is created for the equivalent number of experienced personnel needed for training. It is calculated from the number of new personnel and a parameter for the training overhead percentage. The software development rate is refined again to reduce the effective number of experienced people working directly on development with the new auxiliary:

$$\text{software development rate} = \text{nominal productivity} \cdot [0.8 \cdot \text{new personnel} + 1.2$$
$$\cdot\, (\text{experienced personnel} - \text{experienced personnel needed for training})]$$

Based on studies [Stutzke 1994] and personal experience, 25% of a full-time experienced person's time is initially used for training of new hires. The portion of time expended on training is "released back" once the new people assimilate and become experienced.

First, this formula should be tested with the personnel chain in steady state, which again can be effected by zeroing the assimilation rate or making the delay time inordi-

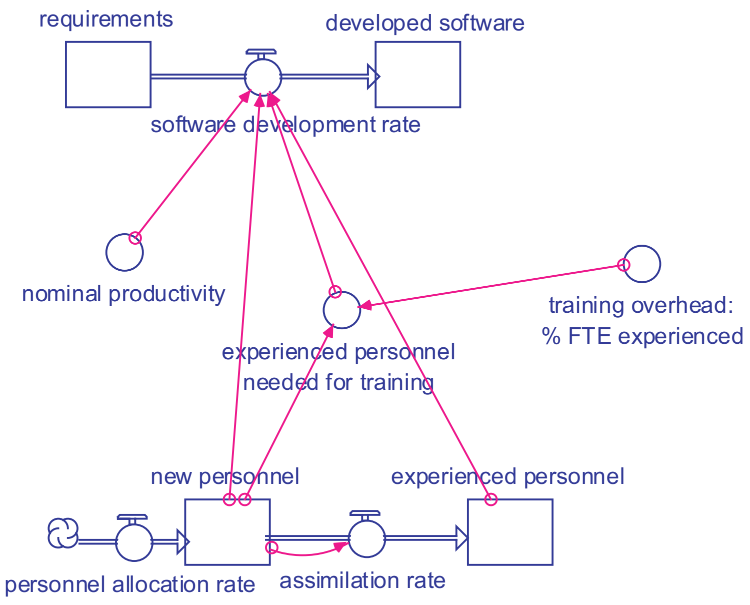

requirements
developed software
software development rate
nominal productivity
training overhead: % FTE experienced
experienced personnel needed for training
new personnel
experienced personnel
personnel allocation rate
assimilation rate

Figure 2.33. Iteration #6 structure.

nately long. The effect is verified in a static situation, assimilation is allowed again, and the results tracked against expected values.

Construction Iteration #7 (Figure 2.34). Account for communication overhead and add the feedback loop to add personnel. A function is defined for communication overhead percentage and used to modify the software development rate. The formula used in the Abdel-Hamid model [Abdel-Hamid, Madnick 1991] is converted into a graph that displays overhead percentage as a function of the number of total personnel. Abdel-Hamid provides references for his formulation, but it is important to note that it only covers a specific region of personnel size (which happens to match our current situation). The lack of scalability will be discussed as a constraint on the model that impacts overall assumptions of team size phenomena. This is due to team repartitioning that occurs when the overall staff increases substantially beyond the nominal size for a single team.

The final software development rate formula adjusts the nominal productivity for communication overhead and becomes

$$\text{software development rate} = \text{nominal productivity} \cdot (1 - \text{communication overhead }\%) \cdot [0.8 \cdot \text{new personnel} + 1.2 \cdot (\text{experienced personnel} - \text{experienced personnel needed for training})]$$

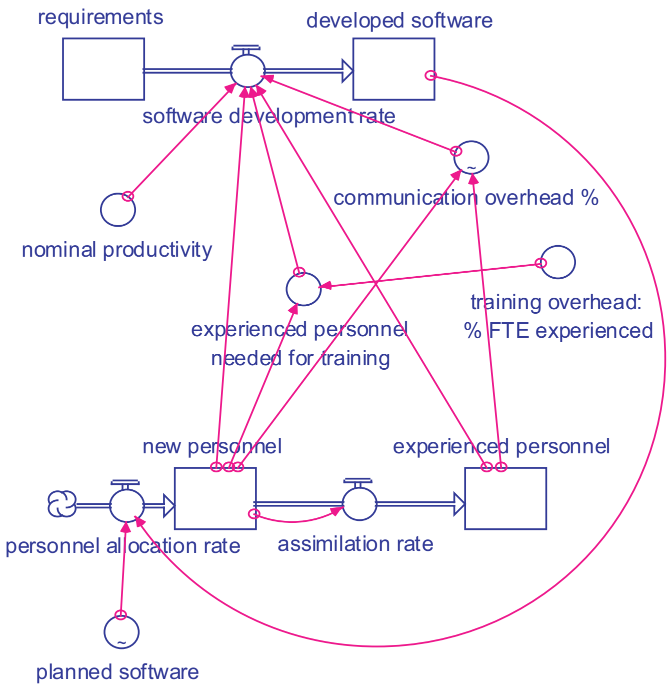


Figure 2.34. Iteration #7 structure.

Like the previous three iterations, this one can be tested first with the personnel chain in steady state before allowing personnel to flow, but in this case the communication overhead percentage should remain the same constant value between runs if the total number of people are equal. The relative proportions of new to experienced personnel do not matter for this effect. The feedback loop should be deactivated at first to keep the personnel chain in steady state, then activate it to induce the dynamics from adding people.

2.7 SIMULATION

Simulation means executing the computer model; some testing also occurs. Run-time parameters must be set, such as the time horizon and any sensitivity parameters. Other run-time options may include whether the model runs in interactive mode or not, plotting and tabulation parameters, the selection of DT (the time step), the form of numerical integration used for internal calculations, and use of input files. When all is ready, you simply start the simulation and let it progress. It will compute the equations as time advances and stop when the time horizon is up or, alternatively, at a user intervention point.

As part of the iterative model building process, it is quite possible (and recommended) to have already simulated early versions of the model during formation. The iterative simulations continue, leading up to fuller validation tests against the progressively complete model.

Simulation inherently entails a certain amount of white-box and other verification testing (just like unit testing is part of coding). Hence, there is some overlap with Section 2.8, Model Assessment. The white-box testing addresses the specific structural components and equations of an elaborated model. Section 2.8 describes further validation tests, many of which resemble black-box testing, whereby the model at large is assessed against the goals of the study, real-world policies, and insights. That assessment builds upon the testing performed in this stage.

Steps in verification testing during initial simulation and subsequent assessment include:

- Eliminating mechanical mistakes
- Ensuring steady-state behavior
- Initial robustness testing with idealized test inputs
- Trying to recreate reference behavior patterns
- Assessing the sensitivity of the base case run to parameter values
- Assessing the model sensitivity against structural and policy changes

In the previous stage of model construction, you should have already verified the dimensional balance of all equations so that units match on both sides. Flows should have the same units of measure as the levels they are attached to with the addition of "per time." Other questionable model elements should also be resolved before simulation testing begins.

During simulation testing, you attempt to ferret out mechanical mistakes. Problems may be due to coding or logic errors. Possible sources include numerical data errors, unexpected random variates (see Appendix A), inconsistent units, entity flow problems (e.g., levels going negative), or incorrect statistics specifications. Some error detection and prevention approaches include:

- Modular validation of the model
- Well-commented and legible model equations
- Trying alternative approaches to crosscheck the results, such as spreadsheets or manual numerical evaluations
- Use of outside analysts to assess the model
- Inducing infrequent events to uncover "hidden" model areas
- Using animations available in simulation tools to visualize the changing rates, levels, and other functions

You should make short model runs, look for anomalous values, and identify mistakes. It is wise to instrument all variables in the model during the formative stages. An example first-level bug at this stage might be that a stock goes negative. Make the process highly iterative by adding a little to the model each time to fix it; keep it "top-down and iterative" as explained in model conceptualization.

During testing, parameter values should be specified with simple, internally consistent numbers instead of spending much effort collecting numerical data. Try to quantify qualitative variables using a consistent scale. A relative scale of 0–1 is extremely useful in this context also. Choose initial numbers so the system can reach a steady-state condition.

Start testing activities with basic debugging of a very short simulation, like for a single DT timestep. Check out and fix any aberrations that you observe. Debugging can be performed in several ways. Anomalous variables and their inputs can be put into a table (or a graph used for debugging). Printing these values in a table at every DT can help identify bugs in the model formulation. Eyeball the results to see which variable might be causing the problem. Repeat these steps until the mechanical mistakes are fixed. Always make notes and save each model run until the debugging stage is over. As is well known in the software process, defects have a way of producing more defects. It is prudent to keep track of the defects found.

One common bug is using too large a value for the DT. Normally, a value of less than one-half of the smallest time constant will ensure that the DT does not influence the model outputs. To be even safer, the DT should be less than one-third of the smallest time constant. If too large of a DT value is used, the behavior may exhibit oscillations, the oscillations may increase over time, or other jerky behavior may be observed.

A long solution interval generates numerical instability. If the interval is too short, unnecessary computations take place that slow it down (this is becoming less of a hindrance as computing power increases).

Validation begins during simulation. Ensure robustness first by attaining steady state, putting in idealized test inputs, and looking at key variables. Determine if the behavior is plausible; fix and iterate. After the initial verification, you graduate to more sophisticated tests described in later sections:

- *Robustness testing*—torture the model to see where it breaks down, such as using steps and pulse inputs to see how well the model regulates itself.
- *Reference behavior testing*—does the model generate reference behaviors and how robust is the pattern to changes in model parameters.
- *Sensitivity analysis testing*—determine how sensitive and robust policy recommendations are to variations in parameters.
- *Scenario analysis testing*—vary the external environment and see if policy recommendations remain robust.
- *Model boundary testing*—challenge the sinks and sources by assessing if policy recommendations would change if the clouds were replaced by levels, and challenge converters by assessing what would happen if they were replaced with levels.

2.7.1 Steady-State Conditions

Ensure steady-state conditions before controlled experiments. For a model to be in steady-state equilibrium, the inflows and outflows must be balanced for each level such that the net of all flows across a level is zero. Two important reasons for steady-state initialization are:

1. Ensuring that model parameters are internally consistent.
2. After achieving steady state, then controlled experiments can be run to further test robustness and policies.

A base steady-state simulation run should be completed before making conclusions about nonsteady-state behavior. The same debugging techniques described in the previous section can also be useful in steady-state testing. Here you determine if any stock values change and find the culprits.

Sometimes, only portions of a model are put into steady state during a particular testing stage. In the Brooks's Law model, for example, we focus on putting the personnel chain into steady-state conditions. If the task development chain was in steady state then no project progress would be made. The main chain for software tasks must flow properly with the personnel chain in steady state (i.e., no personnel changes) before other tests are made.

On the first steady-state run, choose numbers that are simple and make sense with each other. Internally consistent numbers will help put a model into steady state. Sometimes, one can use algebraic initialization to derive initial values for levels. One should be able to calculate equilibrium levels independent of any simulation tools, at least as a crosscheck of steady-state conditions.

Determine how much latitude there is for specifying flow values. Smaller numbers with few digits are easier to understand when assessing the model. It is possible that you will need to modify numbers from an actual system to achieve steady state, because few real systems are in steady state. You can incorporate other real-world data on subsequent passes as appropriate.

2.7.1.1 Example: Steady-State Behavior

A base equilibrium run of the Brooks's Law model is first needed to start assessing its behavior. Complete steady state is achieved when neither software tasks nor personnel are flowing. This is a degenerate case that shows nothing is changing in the model, and is not shown. Steady state conditions relative to personnel levels are present when no additional people are added. These conditions are necessary to validate a model before adding perturbations. Figure 2.35 shows the steady-state behavior relative to personnel for developing 500 initially required function points with a constant staff of 20 experienced personnel. This represents the project with no hiring adjustments.

The final completion time with the default parameters and no additional hiring is 274 days. The graphic results are not very interesting but do provide confidence that the model behaves as expected and perturbations can now be introduced.

2.7.2 Test Functions

Models are tested through controlled experimentation with idealized test inputs to understand their inner workings. Test functions are used to exercise models by introducing disturbances such as a sudden variable increase, a constant signal, a steady decline, oscillation, or randomness. Commonly used test inputs include the pulse, step func-

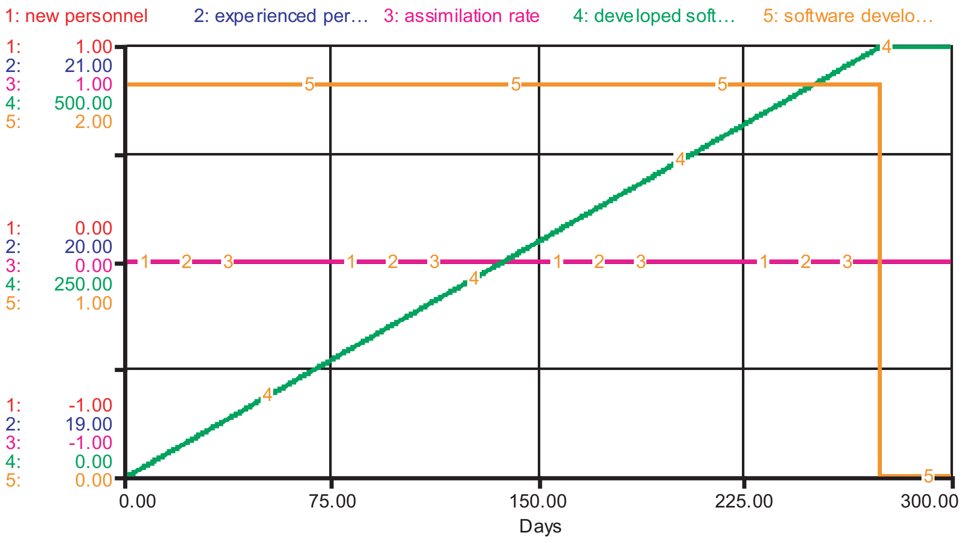


Figure 2.35. Steady-state behavior of Brooks's Law model (personnel in steady state).

tion, and a ramp function. These functions help to expose model relationships and enable understanding of the model and the system it represents. Behavior of pieces of the model can be understood.

Graph the response of key variables to assess how they respond to the disturbances that knock them out of steady state. Determine if the model is showing unreasonable behavior, such as levels going negative, levels continuing to grow, too short of a response time, and so on. You may want to trace the paths through the model to help understand the results. Sometimes, a feedback loop is missing or it is too strong or too weak. A high-frequency oscillation might be caused from using too large a value for DT. A smooth but expanding oscillation pattern might be due to the integration method, and can be fixed by using the Runge–Kutta method instead of Euler's (an option set in the simulation software).

The pulse provides an instantaneous, isolated change in a variable and resembles a spike. The variable returns to its previous value after the momentary change. A pulse was used in the initial Brooks's Law model to effect the abrupt addition of new personnel. In simulation, a pulse occurs over a DT time interval, which is the closest approximation to an instant of time. The change produced in a level by a pulse is the height of the pulse multiplied by the DT value.

The step function changes a quantity abruptly at some point in time and it stays at the new constant level. A combination of step functions can be used to add or subtract constant signals to produce rectangular shapes for variables. A step is useful for idealized testing, such as a first approximation of a staffing profile to understand the steady-state characteristics of a process model.

A ramp is a linear function of time that continuously grows or declines with a specified slope. It can be used to model a gradual increase or decrease in some system variable.

The three idealized test inputs are shown in Figure 2.36. The pulse is set to an impulse quantity of 100 at time = 5. It appears as a spike to 40 because the simulation DT interval is 0.25 time units, so 100/0.25 = 40. The step function begins at time = 10, rises to a value of 10, and stays there. The ramp function has a slope of 1 and begins at time = 15. Figure 2.37 shows the simple equations that implement these.

It is instructive to see how levels react to these standard inputs. If we apply the three test functions in Figure 2.37 as hiring-rate inputs to personnel levels, the corresponding behavior in Figure 2.38 results. See model file *test functions.itm.* The personnel level rises by the pulse magnitude, it linearly increases to the constant step input, and shows exponential growth to the ramp input. These same test inputs could have been used to model behavior of other process variables, such as the influx of software requirements. The level of requirements to be implemented would react in the same fashion to the generic inputs.

Other test functions are readily available in simulation packages such as standard trigonometric functions for oscillating behavior, random number generators, probability distributions, combined logical operations, and more.

As a model is built iteratively in pieces, submodels are constructed and perturbed with test functions before being integrated together. Care must be used when deciding which submodel variables to perturb with test functions and the choice of tests. The

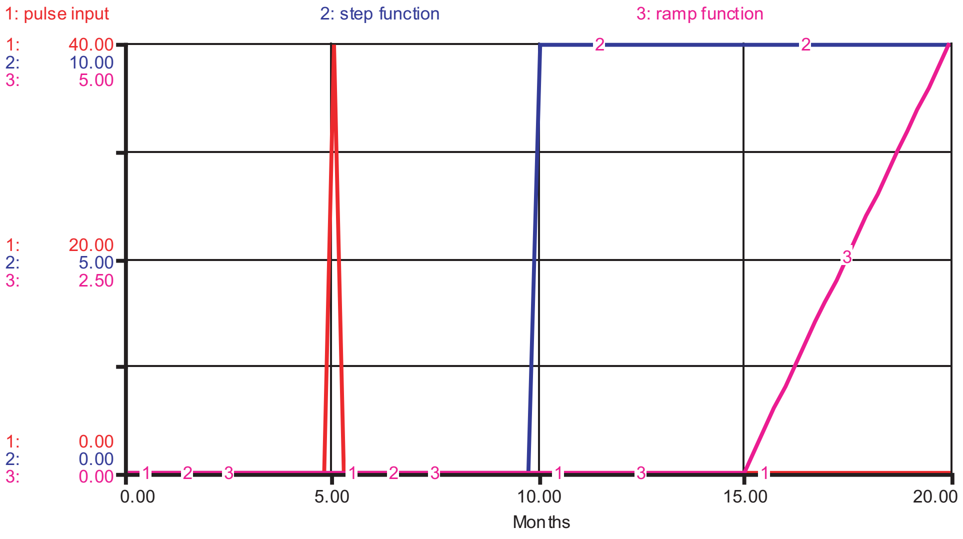


Figure 2.36. Test input functions.

test inputs should go into equations that represent inputs from other model sectors or from outside the model boundary. Common choices for test inputs are rate equations or the auxiliary equations that feed into rates.

2.7.3 Reference Behavior

A model should eventually be checked against reference behavior patterns after assuring that it is free of mechanical bugs and responds to idealized test inputs as expected. Only look for qualitative similarity on the first pass. Eventually, you will want to replicate reference behavior using real-world numbers and check the resulting behavior similarly using historical real-world data.

The introductory graph for productivity from the Brooks's Law model in Chapter 1 compares favorably with the conceptualized Figure 2.13, and Figure 2.39 demonstrates task progress against the reference behavior from Figure 2.14. More applied examples of model reference behavior evaluation are shown in Chapters 4 through 6.

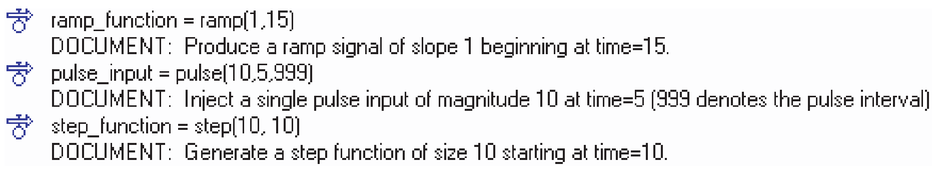

```
ramp_function = ramp(1,15)
DOCUMENT: Produce a ramp signal of slope 1 beginning at time=15.
pulse_input = pulse(10,5,999)
DOCUMENT: Inject a single pulse input of magnitude 10 at time=5 (999 denotes the pulse interval)
step_function = step(10, 10)
DOCUMENT: Generate a step function of size 10 starting at time=10.
```

Figure 2.37. Test input equations.

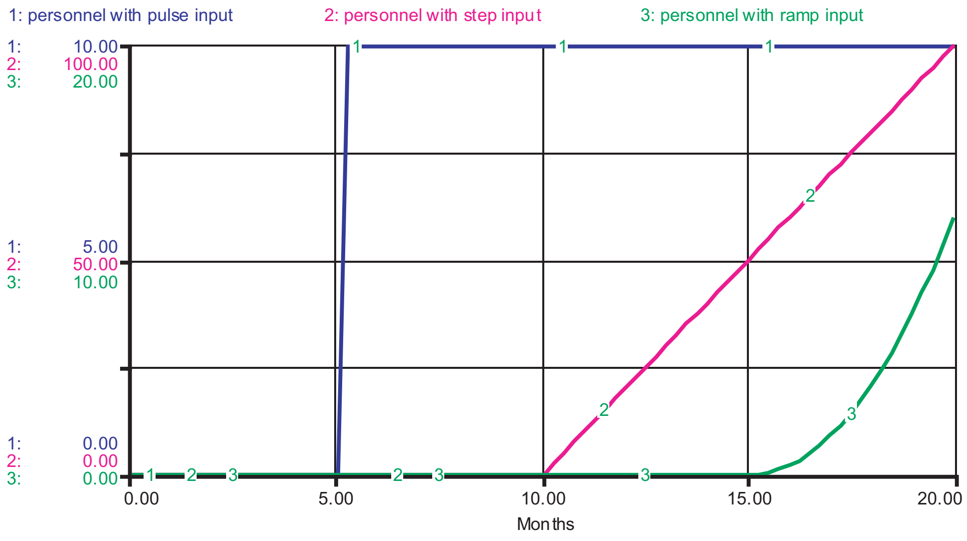


Figure 2.38. Using test inputs as hiring rates to personnel levels.

2.8 MODEL ASSESSMENT

Model assessment is a collection of verification and validation processes, whereby models are iteratively evaluated from several perspectives. During verification, you determine whether the model implementation is error free and properly represents the intended logical behavior (this was largely addressed in the previous section). In validation, you assess if the model helps solve the end user's problem. As always, this must be considered within the context of the study purpose.

Simulation runs should be designed to gain the most insight and understanding from a model. These runs constitute simulation experiments, and a model is essentially a laboratory tool. Plan a sequence of runs that focus on particular structural elements. A hypothesis should be formed before each run. Unexpected results should be thoroughly analyzed. Like other experimental scenarios, a notebook should be kept to record details of model structure, hypotheses, observations, and analysis. Reproducibility of results is required as for other scientific disciplines.

The behavior of each run should be evaluated and reasoned through in terms of the real-world system it represents. A narrative story that sequences through the behavioral dynamics of a run and ties results to the real world is useful for this. Such a story helps bridge the gap between model and reality when real-world terminology is supplemented with model understandings. Explaining model behavior relative to the real world fulfills a goal of modeling—to gain insight and understanding. Ultimately, these insights should increase the probability that improved policies for the real system will be identified and implemented.

Model structure and behavior can be assessed by exploring the behavior due to individual and combined feedback loops. Experiments can be devised to isolate the effects

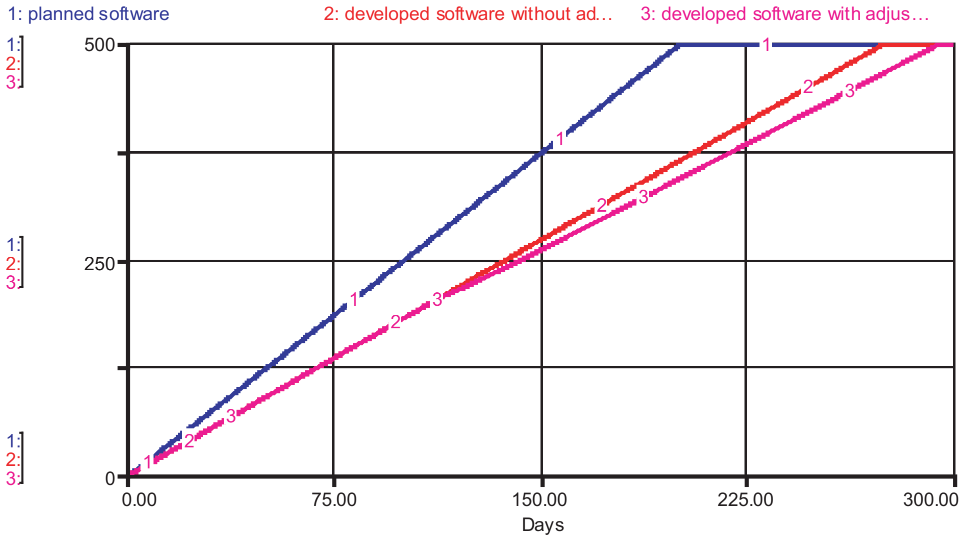


Figure 2.39. Brook's Law model task progress reference behavior.

of specific factors and pinpoint structures responsible for certain behaviors. Due to feedback and circular causality, the notion of linear cause and effect is not very relevant. Thus, feedback structures are sought to explain model behavior as opposed to isolated variables or parameters. Guidance will be given in this section to change parameters and functions, which effectively deactivate chosen feedback loops. With this, the role of a feedback loop can be inferred by analyzing the model behavior with and without the loop.

One method to effectively deactivate loops is to provide extreme parameter values. For example, a rate involving an adjustment time or delay can be zeroed by providing a very large time constant. Consider a general rate equation for goal seeking of the form

$$\text{rate} = (\text{goal} - \text{level})/\text{adjustment time}$$

The rate will be essentially zero when a large enough adjustment time is provided. The feedback loops associated with that rate will be effectively broken. With this type of modification, the structures responsible for behavioral modes can be ascertained by comparing runs in which overlapping model portions are neutralized.

Another way to interrupt the effects of feedback loops is to replace variables, graphs, or table functions with constants so that dynamic relationships are made constant and it is easier to isolate their effects or even break causal relationships.

2.8.1 Model Validation

There is nothing in either the physical or social sciences about which we have perfect information. We can never prove that a model is an exact representation of "reality." Con-

versely, among those things of which we are aware, there is nothing of which we know absolutely nothing. So we always deal with information which is of intermediate quality—it is better than nothing and short of perfection. Models are then to be judged, not on an absolute scale that condemns them for failure to be perfect, but on a relative scale that approves them if they succeed in clarifying our knowledge and our insights into systems.
Jay Forrester [Forrester 1968]

All models are wrong, but some are useful.
George Box [Box 1979]

Stakeholders want to know how much trust can be placed in a model. Model validation is the collection of processes that inspire trust and confidence in a model. The amount of trust involves two issues: (1) what question do we want the model to answer, and (2) how accurate must that answer be? Resolution of these issues ultimately rests on human judgment. By definition, no model is perfect, nor should we expect it to be. As in other scientific disciplines, we cannot guarantee to prove truth but attempt to disprove false hypotheses. The absence of suitability and consistency is probably easier to demonstrate.

Model validity is a relative matter. The usefulness of a mathematical simulation model should be judged in comparison with the mental image or other abstract model that would be used instead [Forrester 1968]. Models are successful if they clarify our knowledge and insights into systems.

Keep in mind that a simulation model addresses a problem not a system, as defined by the set of questions being answered. Only those questions are relevant when placing confidence in a model. The statements of model purpose focus the modeling study and aid in judging the validity of the model results. Thus, they also serve as acceptance criteria. At the end, we seek a shared vision and consensus about the model suitability and consistency. Assuming that the model purpose is improved policy, then the modeler and other stakeholders must create a consensus together.

There is a wide range of model precision in the field. Some models are tightly calibrated to real data, whereas some are not much more than qualitative conjecture, depending on the modeling goals and questions. The nature of the field is that models are produced with varying precision due to diverse modeling purposes.

No single test is enough to validate a system dynamics model. Rather, validation is an ongoing mix of activities in the modeling process with a combination of tests. Approaches for checking validity include parameter and relationship testing, structural and boundary testing, sensitivity analysis, and testing extreme or absurd conditions. This section discusses the multiperspective validation of system dynamics models. Validation is performed with carefully designed test cases, simulation results are compared to existing data, and demonstration of the models for specific purposes should be performed. This includes a number of objective tests. Eventually, insights should be provided by experimenting with the models.

Model validation has been a controversial issue in the software engineering community as system dynamics extends quantitative validation with an extensive range and depth of qualitative criteria. It is akin to a combination of structural and functional testing, and supplemented with multiple quality criteria. The multiple perspectives of quantitative and qualitative validation need to be better understood and accepted by the community. This acceptance can be partially aided by improvements in metrics collection, but software engineers should also gain an appreciation of qualitative validation. The qualitative aspects are sometimes viewed as "soft" validation, but software designers can learn some lessons about judging system quality and ensuring robustness from the system dynamics camp.

There is often confusion with point prediction versus "understanding." In terms of validation, it depends on whether the model purpose is point prediction or behavioral understanding. In many contexts, relative results are more than sufficient as opposed to absolute point predictions of state variables. Knowing the relative differences between policy options is often good enough. Being able to predict absolute quantities is frequently irrelevant.

The emphasis during validation is on building confidence in the suitability of the model for its intended purposes and its consistency with observed reality [Richardson 1991]. Model structure and behavior over time are both examined. A summary of validation tests is shown in Table 2-4, Table 2-5, and Table 2-6, adapted from [Richardson 1981] and [Forrester, Senge 1980]. The passing criteria are identified for each test. The criteria for a valid model consist of the tests in this table and associated subtests. Each individual test is insufficient alone to validate a model, but taken together they provide a robust means of filtering out weak models.

Table 2-4. Model validation tests—suitability for purpose

Focus	Test	Passing Criteria
Structure	Dimensional consistency	Variable dimensions agree with the computation using right units, ensuring that the model is properly balanced
	Extreme conditions in equations	Model equations make sense using extreme values
	Boundary adequacy Important variables Policy levers	Model structure contains variables and feedback effects for purpose of study
Behavior	Parameter (in)sensitivity	
	Behavior characteristics	Model behavior sensitive to reasonable variations in parameters
	Policy conclusions	Policy conclusions sensitive to reasonable variations in parameters
	Structural (in)sensitivity	
	Behavior characteristics	Model behavior sensitive to reasonable alternative structures
	Policy conclusions	Policy conclusions sensitive to reasonable alternative structures

Table 2-5. Model validation tests—consistency with reality

Focus	Test	Passing Criteria
Structure	Face validity Rates and levels Information feedback Delays	Model structure resembles real system to persons familiar with system
	Parameter values Conceptual fit Numerical fit	Parameters recognizable in real system and values are consistent with best available information about real system
Behavior	Replication of reference modes (boundary adequacy for behavior) Problem behavior Past policies Anticipated behavior	Model endogenously reproduces reference behavior modes that initially defined the study, including problematic behavior, observed responses to past policies and conceptually anticipated behavior
	Surprise behavior	Model produces unexpected behavior under certain test conditions: (1) model identifies possible behavior, (2) model is incorrect and must be revised
	Extreme condition simulations	Model behaves well under extreme conditions or policies, showing that formulation is sensible
	Statistical tests Time series analyses Correlation and regression	Model output behaves statistically with real system data; shows same characteristics

The battery of tests in Table 2-4, Table 2-5, and Table 2-6 consider the model's suitability for purpose, consistency with reality, and utility and effectiveness from both structural and behavioral perspectives. For the latter criteria, consider how effective the model is in achieving the purposes of the study. Can the model and its results be used?

Specific modeling objectives should be identified for individual models dependent on the specific application. Simulation test case results are to compared against collected data and other published data, existing theory, and other prediction models. Testing includes examining the ability of the model to generate proper reference behavior, which consists of time histories for all model variables.

When assessing the model consistency to reality (see Table 2-5), it may be a struggle to obtain good metrics representing observed data. This should not be a showstopper unless the expressed purpose is to reproduce past data with a great degree of accuracy and precision. As with other modeling techniques, relative results are often good enough to base decisions on. If desirable, statistical measures like root mean square error (RMSE), relative prediction error, and so on can be applied to the point predictions of simulation models. It should also be pointed out that surprise behavior by a model (see Table 2-6) is not necessarily a bad sign. Counterintuitive results could reveal a new insight and indicate that learning is taking place; old notions are shattered.

The results of judging a model against these individual criteria should be summa-

Table 2-6. Model validation tests—utility and effectiveness of a suitable model

Focus	Test	Passing Criteria
Structure	Appropriateness of model characteristics for audience Size Simplicity/complexity Aggregation/detail	Model simplicity, complexity and size is appropriate for audience
Behavior	Counterintuitive behavior	Model exhibits seemingly counterintuitive behavior in response to some policies, but is eventually seen as implication of real system structure
	Generation of insights	Model is capable of generating new insights about system

rized in appropriate documentation. A good supplemental reference that uses this same framework for verification and validation is [Sterman 2000]. He also provides additional procedures and worked out examples with statistical techniques.

2.8.2 Model Sensitivity Analysis

A model is numerically sensitive if a parameter or structural change results in changes to calculated parameters in a simulation run. Sensitivity analysis is useful in model development, validation, and communicating to others. It helps to build confidence in a model by evaluating its uncertainties. Stakeholders must know the degree to which model analyses and policy recommendations might change with respect to alternative assumptions. For example, if minor and reasonable variations in the model impact conclusions about the real world, then those conclusions are probably not warranted.

Sensitivity analysis also allows the determination of the level of accuracy needed in a parameter for a model to be useful. If a model is found to be insensitive to a chosen parameter, then the modeler knows that an estimate may be used instead of trying to achieve maximum precision. All too often, parameters are difficult or even impossible to measure in a real software process.

Sensitivity analysis will also help determine which parameter values are reasonable to use. If model behavior matches real-world observations, then the values used may reflect real-world data. Sensitivity tests also support system understanding, since experimentation can provide insights into system behavior under extreme conditions. Ultimately, it is desired to determine the high-leverage parameters for which the system is highly sensitive to. Those parameters that significantly influence system behavior represent leverage points in the real system.

There are different types of sensitivity analysis applied to simulation models:

- *Numerical.* The sensitivity of computed numerical values to changes in parameters or structural model changes.
- *Behavioral.* The degree to which model behavior changes when a parameter is changed or an alternate formulation is used.

- *Policy.* The degree to which policy conclusions change with respect to reasonable model changes across a range of reasonable values.

Numerical or parameter sensitivity analysis looks at factor setting, which includes varying a constant between runs. Graphs and table functions can also be modified over a range of meaningful alternative formulations to test sensitivity. The effects of different functions should be evaluated just like the effects of parameter changes. Structural sensitivity analysis looks at model structures—the linking of rates, levels, and loops. For example, should a flowing entity be represented with a single aggregate level or with more intermediate levels. Behavioral sensitivity is observed when the graphs over time change with parameter changes. Policy sensitivity is potentially the most damaging sensitivity because policies are the desired end result of a modeling study. Ultimately, policy insensitivity is the type that really counts.

If the assumptions about structure change, then the conclusions from the model will almost certainly change. Drastic structural changes include redefining the model boundary. Parameter values generally do not have as much of an effect on behavior as do feedback loops, rates, and level structures. This is one characteristic of system dynamics models with dominant feedback loops that are primarily responsible for model behavior. Another reason that such multiloop models often exhibit parameter insensitivity is due to compensating feedback loops. While one feedback loop gets stronger or weaker with a parameter change, other loops in the overall system may strengthen or diminish to compensate.

The compensating feedback is typical of man-made systems that were expressly designed to be insensitive to parameter changes (consider a governor device or car cruise control that maintains a constant speed with changing conditions). It is also instructive to realize that societal systems like software project environments have "built-in" compensating feedback that tends to resist policies designed to improve behavior. This is frequently seen in backlash behavior to process improvement initiatives.

Sometimes, parameters will have negligible effect on behavior or none at all. Also, given that *relative* results from a model are frequently sufficient for policy conclusions, then precise parameter assignations are not always worth the extra effort. If the patterns remain consistent as a parameter changes over a feasible range, then that is often enough to know without having to be more precise with the parameter values.

Improved policies are ultimately desired. If policies are sensitive, investigations should be made to determine if the sensitivity is an artifact of the model or an accurate reflection of the real system being studied. If the model structure is deemed to be consistent with the real system, then the parameter that causes the sensitivity should be estimated as accurately as possible. It might also be wise to seek other advantageous policies that do not depend so heavily on the value of a particular parameter.

Conversely, if policy conclusions remain solid as parameters are varied over reasonable ranges, then those parameters do not need to be estimated with any more accuracy than the variable ranges used.

2.8.2.1 Example: Sensitivity Analysis

This section demonstrates sensitivity analysis applied to the simple Brooks's Law model. It will explore the effects of various parameter changes, initial value changes, and minor structural changes on the system behavior. The sensitivity of the model to different changes is evaluated by running a series of tests.

The first parameter change is the personnel allocation rate. Figure 2.40 (shown previously in Chapter 1) shows the effect on software development rate of adding five and ten people instantaneously at time = 110 days. The graph shows reasonable behavior for the model.

What if hiring takes the form of a step increase rather than a pulse? Figure 2.41 shows the model runs for step increases of 1 person for 10 days and 0.5 persons for 20 days (anything shorter will largely resemble the pulse). The first trendline is the pulse of 10 people, the second is the 10 day step increase, and the third is the 20 day step increase.

Another parameter to vary is the assimilation rate. Values of 20, 40, and 60 days will be used (these represent 1, 2, and 3 working months, respectively). Since the base run has no personnel allocation, these changes will have no impact to it. The case of allocating 10 more people at day 110 with an instant pulse will be used to examine the impact. Figure 2.42 shows sensitivity results.

The behavior is as expected when the training assimilation takes longer. The schedule time varies slightly from 271 days to 276 days when the assimilation increases from 20 to 60 days. Thus, the schedule time is relatively insensitive to these changes.

In conclusion, there are several possible responses when a model is shown to be sensitive. If parameters or graph functions have significant impact, then they should be

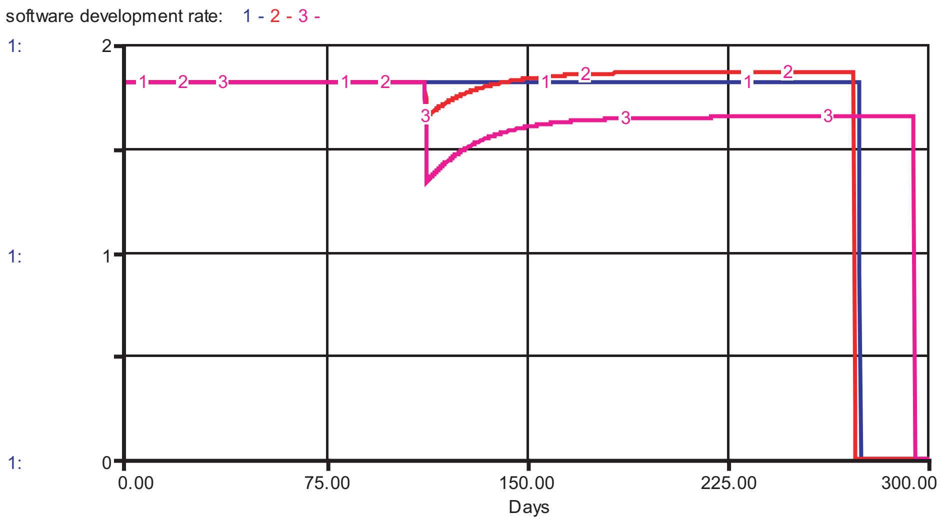


Figure 2.40. Sensitivity of software development rate to personnel allocation pulses (1: no extra hiring, 2: add 5 people on 110th day, 3: add 10 people on 110th day).

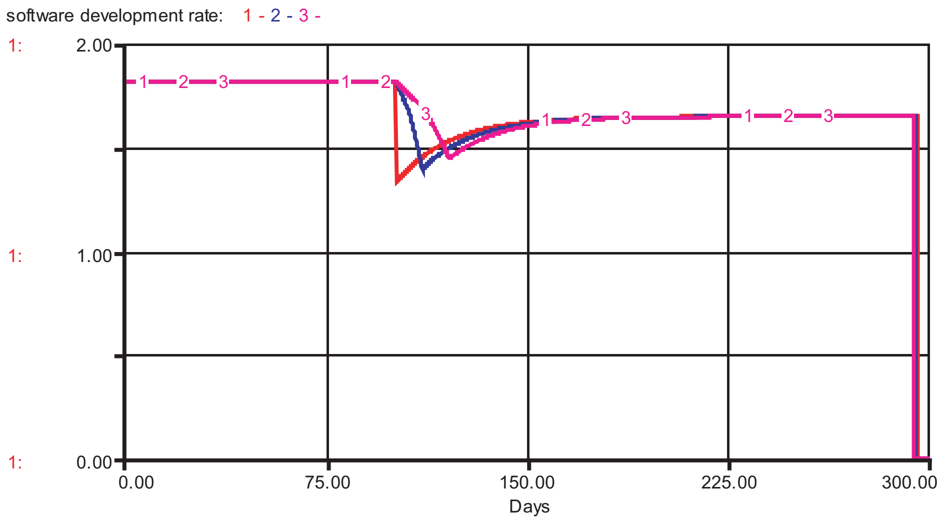


Figure 2.41. Sensitivity of software development rate to personnel allocation steps (1: add 10 people on 110th day, 2: step of 1 new person for 10 days, 3: step of 0.5 new person for 20 days).

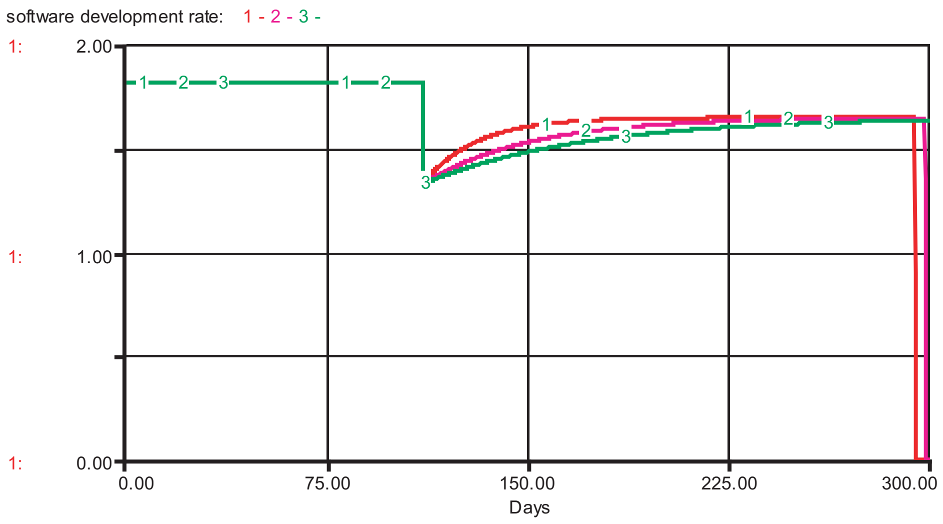


Figure 2.42. Sensitivity of software development rate to different assimilation times (1: assimilation = 20 days, 2: assimilation = 40 days, 3: assimilation = 60 days).

estimated with much care. Another response is to disaggregate and reformulate the model with more detail. Possibly, a parameter or graph covers multiple effects that should be broken out separately. Finally, the sensitivity can be interpreted to identify an important leverage point in the system. Thus, intervention in the real process will truly have a significant impact.

2.8.3 Monte Carlo Analysis

Monte Carlo analysis is a "game of chance" technique used to assess model outcome ranges by applying random sampling. Samples are taken from known input probability distributions to create output distributions. It estimates the likely range of outcomes from a complex random process by simulating the process a large number of times. The following steps are performed for *n* iterations in a Monte Carlo analysis, where an iteration refers to a single simulation run:

1. For each random variable, take a sample from its probability distribution function and calculate its value.
2. Run a simulation using the random input samples and compute the corresponding simulation outputs.
3. Repeat the above steps until *n* simulation runs are performed.
4. Determine the output probability distributions of selected dependent variables using the *n* values from the runs.

See Appendix A for additional details on Monte Carlo analysis, including the following example, which is a shortened version from the appendix.

2.8.3.1 Example: Monte Carlo Analysis

This example will simulate the randomness of the size input to a dynamic effort model, and quantify the resulting output in probabilistic terms. Assume that the likely values for size can be represented with a normal probability distribution with a mean of 50 KSLOC and a standard deviation of 10 KSLOC.

To implement Monte Carlo analysis, a small set of $n = 16$ random samples will be generated for size input to the Dynamic COCOMO model. In actual practice, 16 would be a very low number (except for very large and expensive simulation iterations). Generally 50–100 iterations should be considered as a minimum for filling out distributions, and up to 1000 iterations will give good results in many situations.

First, 16 random numbers between 0 and 1 are generated. The inverse transform technique is then used to determine size by mapping the random numbers onto the cumulative distribution function. Appendix A shows full details of this example, including the set of 16 random numbers (r_i) and the generated size values $F^{-1}(r_i)$, and graphically illustrates the inverse transform technique using the same numbers to determine size 16 times via the cumulative distribution function.

Dynamic COCOMO is then run 16 times with the respective size inputs. The simulation outputs for the personnel staffing curve in Figure 2.43 demonstrate the model

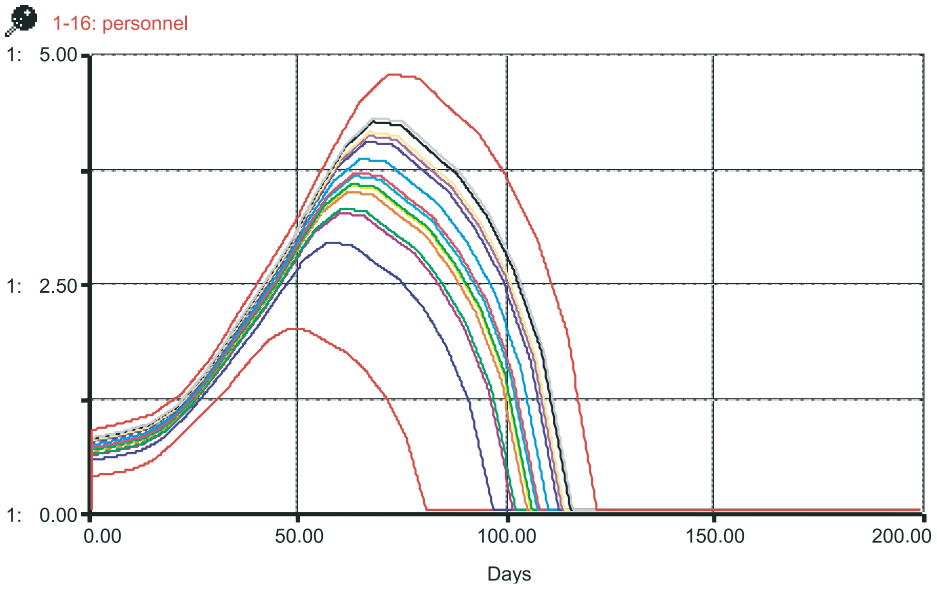


Figure 2.43. Monte Carlo results: personnel staffing curves for 16 runs.

sensitivity to the size samples. Project effort is the area under a staffing curve. Figure 2.44 shows the Monte Carlo results in terms of an effort distribution histogram and its continuous representation. It stacks up the 16 simulation outputs for the total project effort into respective effort bins. A smoother distribution would be seen to be filled out if more iterations were run and plotted beyond the small sample size of 16.

2.9 POLICY ANALYSIS

Policy analysis is the model-based evaluation of process change options, that is, process improvement initiatives. In this stage, we seek to understand what process changes work and why. Investigations are undertaken to determine why particular policies have the effects they do, and to identify policies that can be implemented to improve the real system or process.

Testing of policies and their sensitivities in different scenarios is critical in model evaluation. You alter one parameter at a time at first. Always be conscious of reality constraints on the numbers and policies. Use sensitivity results to identify worthwhile policy initiatives, test the robustness of each policy in isolation, then explore with other parameter changes. Test the policies under different scenarios. Finally, the eventual goal is to update the software process policies based on the analysis results.

However, recommendations from a well-done study should also be explainable and defendable without resorting to a formal model. The final recommendations come from not only manipulating the model but from additional understanding about the real

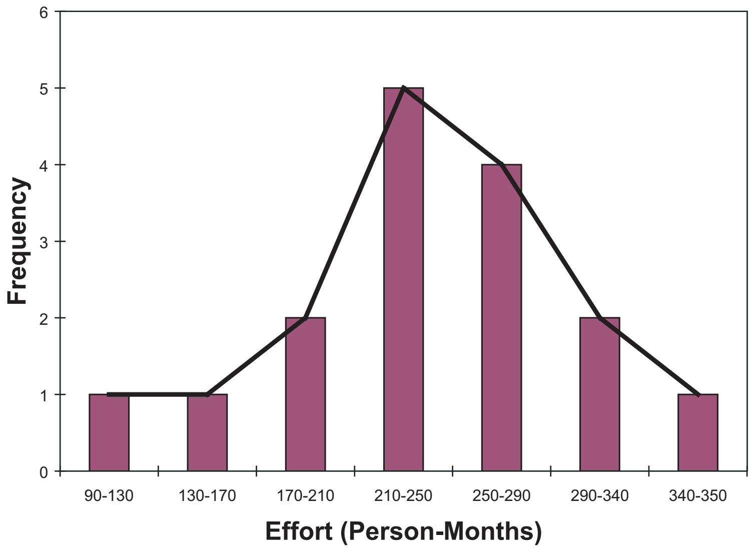


Figure 2.44. Monte Carlo results: effort distribution.

system that was gained during the modeling exercise iterations. The final result of policy analysis is identifying policies that are actually implemented on projects.

The term *policy analysis* has long been used in the system dynamics community. In software process improvement, processes are typically defined by a combination of policies, procedures, guidelines, and standards. A policy in this context generally states what *has* to be done, and the rest of the artifacts provide details of *how.* For this section, these process artifacts are lumped under the umbrella of "policies."

A policy represents a decision on how organizational processes should be executed. An example process policy is that peer reviews must be performed. This policy can be revised and elaborated from modeling insights in terms of what types of peer reviews work best, how frequently, on what artifacts, and so on.

Policy alternatives in the real world correspond to either model parameter changes or structural changes. One always should look for high-leverage change opportunities. Policy parameters with high sensitivity suggest points of leverage in the real system. Changes involving new feedback structure suggest new ways of manipulating information in the real system to improve behavior. The following sections derived from [Richardson, Pugh 1981] discuss parameter changes and structural changes as they relate to policy analysis.

2.9.1 Policy Parameter Changes

Policy changes are often reflected as model parameter changes. Testing a parameter's sensitivity involves making a base run, changing the parameter value, rerunning the simulation, and comparing the resulting model behavior with the base run. Parameters are also sometimes changed in midrun to determine their effects during the course of a simulation. These changes can be interpreted as testing a real-world policy change as long as the change is feasible in the real process. Parameters are classified as policy parameters if their values are within the span of control of people in the real process, or at least controllable to some extent.

A fundamental understanding is necessary about *why* policies have their effects, because it is not sufficient to just know that a particular policy improves model behavior. Understanding of the policy phenomena is compared to what is known or believed about the real system. Then a policy analysis has a better chance of leading to real process implementation decisions.

Much thought and repeated simulations may be necessary to produce a complete understanding in a complex model. One method to understand policy improvements involves deactivating pieces of the model structure until the policy run behaves no differently from the base run with the same pieces deactivated. Graph or table functions should be looked at carefully since they frequently represent assumptions about existing policies.

There are some things to be careful of when analyzing policy changes in a model. For example, one must consider what else might change in the real system when the policy is implemented. There may be other dynamic consequences not yet accounted for. The essence of the real-world policy must be accurately interpreted in terms of the model, to the extent that model aggregation allows. In a sense, the modeler becomes part of the system to accurately simulate a policy change. He/she acts as missing feedback links to manually change other parameters to simulate the side effects of the primary policy change under investigation.

One should also be careful to consider the real limits on policy parameters. A combination of policy parameter changes that work well in a model may be impossible to implement in the real system. The modeler must try to keep policy parameter changes within achievable and realistic bounds (except when testing extreme conditions to gain model insight). Keep in mind that a model does not test what can be implemented, only what is likely to happen if a policy is implemented.

Sensitive policy parameters should be identified, as they indicate possible leverage points in the process. Intimate familiarity with model behavior under different scenarios is critical to locate sensitive parameters. Also important is the location of feedback loops which dominate behavior at various times. Behavior is unlikely to be sensitive to a parameter that has no influence on any dominant loops. Identifying parameters that influence positive loops may also be very useful because such loops are usually destabilizing.

Feedback systems tend to compensate internally for parameter changes, so model behavior in general is insensitive to such changes. So policies represented as parameter changes frequently are not very effective in system dynamics analyses. More dramatic results are usually observed with changes in the policy structure as opposed to parameters.

2.9.2 Policy Structural Changes

New policies often alter or extend the system's feedback structure. The structure of a system with feedback tends to be the strongest single determinant of its time-based behavior. Thus, improvements frequently involve adding new feedback links representing new, beneficial ways of manipulating information about the system.

A simple example of a policy that adds feedback in the process is having to use past project performance and costs to calibrate a new project estimate. Without the usage of historical data to calibrate cost estimation, there is no connection to past performance and, thus, project planning runs open loop. The policy decision to use past data will likely improve the accuracy of new estimates and lead to reduced project overruns.

The main guidance for discovering high-leverage structural changes comes from familiarity with the real system and model. Other principles to keep in mind are that adding a positive feedback loop could have a strong destabilizing influence, and negative loops can add stability. Recall that a negative loop in an oscillatory structure has a damping effect.

A good policy set will be a mix of policies that work in harmony, counteracting the tendencies of undesirable system behavior. Advantageous policies will come to light via intimate knowledge of the system being studied, by insights gleaned from the modeling process, and by considerations of what is possible to implement in the real system.

2.9.3 Policy Validity and Robustness

The ultimate goal of modeling is the application of insights to real-world problems, in such a way that model building will help people manage software processes better. Improved decisions can be made when the dynamic consequences are known. The policy analysis effort and inherent uncertainty of prediction forces one to address the validity of the recommendations and how implementable they are. But people may differ about what constitutes process improvement.

It is the nature of feedback systems, or any complex man-made system, that trade-offs are made. One well-known example in software development is the trade-off of increased cost to accelerate schedule. But how are such trade-offs evaluated? A system dynamics model does not set or evaluate the criteria of improved system behavior. Only people can make such value judgments. A model cannot determine a desirable scenario, but supports people in their weighing of alternatives.

Policy recommendations, like models themselves, are subject to validity and robustness questions. Even if a model has successfully passed all validation tests, how much confidence should be placed in the policy recommendations derived from it?

Robustness refers to the extent to which the real system can deviate from the model assumptions without invalidating policy recommendations based upon it. A recommendation is robust only if it remains advantageous in spite of parameter variations, different exogenous conditions, and reasonable alternatives in model formulation. A policy that is sensitive to such variations is suspect.

Since the real problem will always have aspects that are not captured by a model, a policy must be robust to have real-world effects similar to its effects in a model. For example, few practitioners would trust a policy that flip-flops as a parameter varies over a reasonable range. All policy recommendations should pass rigorous sensitivity tests before being passed on for implementation.

The success of a particular policy should also be tested under a variety of external circumstances. In summary, to build a robust set of policies one must spend considerable effort trying to prove they are not robust. This is like attempting to disprove a proposed theory to see if it still stands up under scrutiny.

2.9.4 Policy Suitability and Feasibility

Robustness is one aspect of policy validity; another is the adequacy of the model to support the recommendations based upon it. Recall that long before policy analysis, it is extremely important to set the model boundary appropriately for the problem being addressed and the questions being asked. During policy analysis, these same boundary questions are addressed. It should be asked whether the factors initially left outside of the boundary could invalidate the model-based analyses.

An important consideration before implementation is whether those responsible for policy in the real system can be convinced of the value of the model-based recommendations. Another is how is the real system likely to respond to the process of implementation?

People responsible for managing complex human systems base their policies on their own mental models. Thus, the policy recommendations must be formulated and argued so as to fit into their mental models. The reasons behind model behavior must be made plain and clear to the process owners.

This is more justification that all stakeholders be integral to the modeling process, and that the WinWin spiral lifecycle is a good fit for modeling. Insights are more likely to come out of the process rather than the deliverable reports of a modeling study. The likelihood of results being implemented are seriously diminished if the process participants are not involved.

In summary, successful implementation of new processes depends on good policies, acceptance by managers and process owners, as well as a sensitively planned transition path to the new process(es). It should also be pointed out that the process of implementing policy change is a feedback problem in and of itself. This problem is left as research for interested readers.

2.9.5 Example: Policy Analysis

Policies contained in the Brooks's Law model must be evaluated with respect to project goals and constraints. Since this is a hypothetical example, we will just list some alternative policies. Options to consider include

- Changing the personnel allocation control rules
- Different timing for adding new people
- The way they are added
- The type of people added
- The team makeup

The correction could take place earlier and/or be a more gradual and continuous approach. Personnel do not have to be added in a single pulse (see the preceding sensitivity tests for this) and, in fact, a pulse was used just for simplicity's sake. More than likely, a real project would ramp up the staff incrementally. The people added do not necessarily have to be new, or they can be able to assimilate faster. Possibly, one could devote support people to training instead of having experienced job performers do the training, which would lessen the training overhead impact. Lastly, the team does not have to stay as a single team, as it could optionally be partitioned.

There might be other organizational considerations to be addressed. Despite the lessened project execution capability, adding people to a project could be a desired policy for other reasons. For example, if training of new hires is important, that alone might be sufficient justification to add people. Possibly, the next project is more critical than the current, and it's more important to be prepared for the upcoming project instead of saving a few days of schedule on the current one.

2.10 CONTINUOUS MODEL IMPROVEMENT

Model reassessment and improvement should be ongoing as part of the iterative development process. Parts of a model are reformulated and refined (sometimes deleted), and new parts added during iteration. The iteration will help ensure that the model is consistent with reality, suited well for its expressed purposes, and understood. Sometimes, constants are reformulated into variables, levels are disaggregated into more levels, new feedback structure is added, or even new hypotheses about dynamic behavior are developed. This section will discuss various issues in the ongoing refinement process.

Continually challenge the work you have already done during the course of a modeling project. It is possible that the original problem will have morphed and/or some of the underlying assumptions do not hold anymore. Take a fresh critical look at the model, and iterate the entire process. Sit back, ponder, and do sanity checks against the known reality. Some starting points to investigate include:

- Challenge what has been left out. Look at sources and sinks at the beginning and end of chains, and consider feedback and policy implications of replacing them with levels.
- Challenge what has been put in. Is it too much or too little detail? It all depends on the purpose. Look at the levels and converters to see if they are commensurate with the model purpose and available data.
- Does the model contain the measures of policy effectiveness that are used in the real process? If not, then these important measurement criteria commensurate with real decision making should be included.

If a full cycle has been completed through policy analysis and implementation, then it is time to plan the next cycle of model improvement. Per the WinWin spiral model, review the previous model development and decide on the commitment for the next phase based on the latest objectives, constraints, and alternatives.

2.10.1 Disaggregation

Sometimes, levels should be disaggregated into more than one level. The decision to do so should be based on policy analysis and/or model behavior. Disaggregation is called for if a policy issue cannot be addressed otherwise. For example, if staffing considerations must account for the different levels of new and experienced workers, then a single aggregate level for personnel should be split. Another reason is when disaggregation will significantly change the behavior of a model. In the Brooks's Law model, for example, we would not get the same dynamic behavior if new and experienced personnel were not tracked separately. The differing productivities and training overhead would not come into play.

In the Brooks's Law model and other studies involving personnel training or skill development, new policy levers are available when multiple personnel levels are used to represent different experience or skill levels. Implementation-wise, splitting a level into two may frequently double the number of equations associated with the original level. This is a potential pitfall to be aware of.

It should be stressed that disaggregating just to better model the real system is not always warranted. It should be done only if the resulting behavior change is important and particularly if policy conclusions depend on it. Otherwise, adding more levels than required is distracting (visually and conceptually) and can make a model harder to understand.

2.10.2 Feedback Loops

Each refinement to a model complicates things. If additional feedback loops are to be added to an existing model, it is highly advisable to add them sequentially instead of all at once. Otherwise, a compounding of changes may lead to loss of model control. By adding one new feedback link at a time, it is easier to identify and fix model defects (like the "code a little, test a little" mantra).

2.10.3 Hypotheses

With a simulation model, one can incorporate more than one hypothesis and compare their potential to recreate system behavior. As part of the iterative approach, a single dynamic hypothesis in the beginning will simplify things. After assessing the model against the original hypothesis, in subsequent iterations alternates can be added that may also be responsible for behavior.

When refining a model, the same techniques are used as during the original model development. Plot a lot and study the outputs. Make sure important key variables and feedback effects are included on the plots. The plots should be kept fairly simple without too many variables on one graph. Here the rule of "7 ± 2" also holds. Often, the scale of plots can be fixed so to that multiple runs can be easily compared to each other. Sometimes, tabular outputs are helpful to read more detail.

2.10.4 When to Stop?

There are no definitive rules about precisely when to stop ongoing model refinement. Generally, you can stop if the model has reached a point at which it is well suited for its stated purpose(s) and consistent with reality. As with many processes, a point of diminishing returns usually exists. If the extra benefits of more refinements are outweighed by the additional costs of further refinement, then a point of diminishing returns has been reached and the exercise should be stopped. If the present model is useful for people to generate insights about the problem, then stop. If, on the other hand, the model has increased in size and complexity so that it is hard or impossible to understand, then some retrenchment might be in order. It is quite possible that you have gone further or faster than necessary, so drop back a little.

Knowing when to stop thus requires some intuition and wisdom. It is certainly tempting to keep going, but always keep in mind that no model is 100% complete and perfect. The point of stopping is also tied to the question of model validity described in Section 2.8.1.

2.10.5 Example: Model Improvement Next Steps

The Brooks's Law model described so far is based on simplified assumptions and has certain limitations. The following enhancements would improve usage of the model:

- Add a stop to the simulation when all requirements are developed. This will prevent the model from running overtime.
- Make the model scalable for larger team sizes up to 60 people to overcome the current restriction on maximum team size.
- Make a continuous personnel allocation instead of a one-time correction.
- Add the effects of team partitioning.

Figure 2.45 shows the default function for communication overhead that stops at 30 people. This auxiliary relates "communication overhead" to the total number of personnel. If we blindly continue with the formula,

$$\text{communication overhead} = 0.06 \cdot \text{team size}$$

then the equation quickly becomes greater than 100%. What is wrong with this? It does not account for partitioning into more teams as the total number of people grows. It assumes that a single team will remain and continue to grow indefinitely in size without breaking into more teams.

Brooks stated that adding people to a software project increases the effort in three ways: training new people, added intercommunication, and the work and disruption of

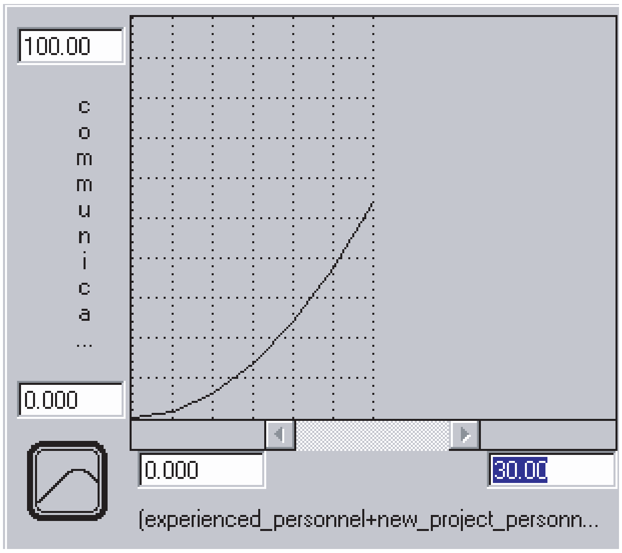


Figure 2.45. Default communication overhead function (communication overhead % versus experienced personnel + new personnel).

team partitioning. We purposely did not address partitioning yet due to its difficult complexities. See Chapter 5 for exercises to enhance the Brooks's Law model to include the effects of partitioning.

2.11 SOFTWARE METRICS CONSIDERATIONS

Process modeling may involve a good deal of metrics collection and analysis, depending on the goals and constraints. When feasible, models should be bolstered by empirical data to reflect reality as closely as possible. However, detailed metrics data is not always warranted or immediately available. The following two sections discuss data collection and describe a useful metrics framework for modeling.

2.11.1 Data Collection

A mathematical model should be based on the best information that is readily available, but the design of a model should not be postponed until all pertinent parameters have been accurately measured. That day will never come.

Jay W. Forrester, *Industrial Dynamics* [Forrester 1968]

Existing data often does not completely match the requirements for a new modeling effort. Comprehensive process models cover a number of different aspects, and these models are best quantified using a single data source commensurate with the comprehensive nature of the model. However, very few published empirical investigations cover a wide range of process aspects with solid statistics. Studies usually have a narrower focus. Fine-grained models should have commensurately detailed data for parameterization. To ensure that the data fits the model, the metrics defined for data collection should be the same as those used in the model.

Having quality data to support modeling efforts is one of the most difficult aspects. Accurate simulation results depend on accurate parameter values, and calibration depends on accurate measurements from the process. Whenever possible, available metrics should be used to quantify relationships between factors. However, not having data at hand should not seriously impair model development. There may be a use for judgmental data and management opinion to estimate parameters for "soft variables" for which numerical data is lacking. Experts can make educated guesses.

Some ways to cope with lack of data from [Kellner, Raffo 1997] include:

- Adjust existing values to approximate desired variables
- Construct values from other detailed records
- Obtain estimates from personnel involved
- Use typical values taken from the literature

One must often go on an archeological data dig. For example, obtaining original inspection data sheets may provide critical data details not available from summary records. Field data to drive models can also sometimes be artificially generated.

The rigor used for collecting data should tie back to the modeling goals. Not every study needs highly precise, validated metrics. Often, qualitative or relative results are good enough for the purpose of a study. The level of rigor depends on how the data is going to be used. A goal-directed technique described next is valuable for defining measurements to be used in modeling efforts.

2.11.2 Goal–Question–Metric Framework

The Goal–Question–Metric (GQM) framework [Basili 1992] for conducting metric analyses can be highly useful for simulation modeling. It provides a framework for developing a metrics program and helps ensure that software metrics are mapped to goals. Organizational goals are identified, questions are developed to determine whether the goals are being met, and metrics are identified that can help answer the questions. The framework was developed at the University of Maryland as a mechanism for formalizing the tasks of characterization, planning, construction, analysis, learning, and feedback. The GQM paradigm was developed for all types of studies, particularly studies concerned with improvement issues. The paradigm does not provide specific goals but rather a framework for stating goals and refining them into questions to provide a specification for the data needed to help achieve the goals.

GQM consists of three primary steps:

1. Generate a set of organizational goals.
2. Derive a set of questions relating to the goals.
3. Develop a set of metrics needed to answer the questions.

The goals are based upon the needs of the organization, and they help in determining whether or not you improved what you wanted to. Goals are defined in terms of purpose, perspective, and environment using the generic templates as follows:

Purpose. {To characterize, evaluate, predict, or motivate} {the process, product, model, or metric} in order to {understand, assess, manage, engineer, learn, or improve it}.

Perspective. Examine the {cost, effectiveness, correctness, defects, changes, product metrics, or reliability} from the point of view of the {developer, manager, customer, or corporate perspective}.

Environment. The environment consists of the following: process factors, people factors, problem factors, methods, tools, and constraints

The questions quantify the goals as completely as possible within the context of the development environment. Questions are classified as product-related or process-related and provide feedback from the quality perspective. Product-related questions define the product and the evaluation of the product with respect to a particular quality (e.g., reliability, user satisfaction). Process-related questions include the quality of use, domain of use, effort of use, effect of use, and feedback from use.

Finally, a set of metrics is developed that provides the information to answer the questions. The actual data needed to answer the questions are identified and associated with each of the questions. As data items are identified, it must be understood how valid the data item will be with respect to accuracy and how well it responds to the specific question. The metrics should have interpretation guidelines, that is, what value of the metric specifies the product's higher quality. Generally, a single metric will not answer a question; a combination of metrics is needed. Figure 2.46 is sample GQM analysis from [Madachy 1995a] for analyzing the inspection process.

2.11.3 Integrated Measurement and Simulation

Measurement efforts and process simulation should be tightly integrated; otherwise, there is a risk of existing data not being commensurate with model constructs (granularity, abstraction level, etc.). There may be unnecessary effort in generation, collection, and possible reformatting of data for simulation models. Effort to locate and assess relevance of empirical data from external sources may be prohibitive.

The GQM framework can help alleviate some of the problems with empirical data. The risk-driven WinWin Spiral model (traditionally used for software projects) is an-

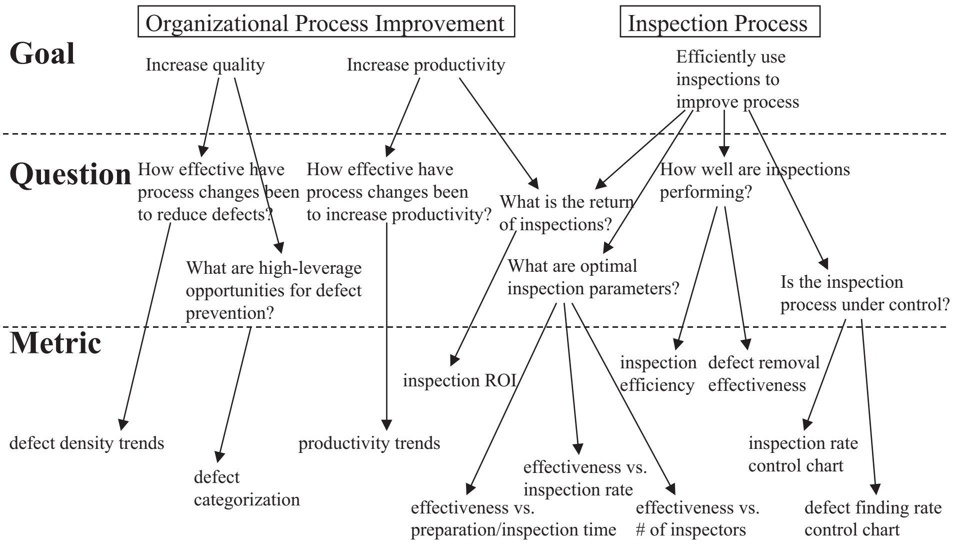

Organizational Process Improvement
Inspection Process
Goal
Increase quality
Increase productivity
Efficiently use inspections to improve process
Question
How effective have process changes been to reduce defects?
How effective have process changes been to increase productivity?
What is the return of inspections?
How well are inspections performing?
What are high-leverage opportunities for defect prevention?
What are optimal inspection parameters?
Is the inspection process under control?
Metric
inspection ROI
inspection efficiency
defect removal effectiveness
defect density trends
productivity trends
inspection rate control chart
defect categorization
effectiveness vs. inspection rate
effectiveness vs. preparation/inspection time
effectiveness vs. # of inspectors
defect finding rate control chart

Figure 2.46. GQM applied to inspection analysis.

other framework applicable to modeling studies and measurement experiments. For example, based on modeling risk considerations, a spiral cycle might be dedicated to a measurement exercise before further elaboration of a software process model (see the example earlier in this chapter).

For a simulation study, the GQM goals must match those of the model stakeholders. The metrics are the key variables that the model will output in order to answer the questions. System dynamics may impose data collection requirements and the GQM software metrics may be used to define reference modes for simulation models. Pfahl and Ruhe have developed some integrated approaches in which the GQM method has been interpreted specifically for the context of process simulation [Pfahl 2001], [Pfahl et al. 2003], [Pfahl, Ruhe 2005].

Goal-oriented measurement plus system dynamics are described as complementary approaches in [Pfahl, Ruhe 2005]. "Dynamic GQM" is an evolutionary and learning-based approach for process modeling. The different process steps in GQM and the system dynamics method are multiplexed together in various ways for different purposes. For example, GQM-based measurement can be used at various stages of the system dynamics modeling process.

Dynamic GQM has several advantages. Whereas GQM is local when one goal is looked at in isolation, system dynamics can capture effects on a global level. GQM is static and does not reflect changes over time. Dynamic GQM integrates and synchronizes individual GQM models from a global system perspective. It incrementally increases the validity of both GQM and system dynamics models by mutually reusing and validating the models. Finally, it is a means to integrate real-world empirical results with simulation experiments.

More detailed guidance beyond Dynamic GQM is provided in [Pfahl, Ruhe 2005] where Integrated Measurement, Modelling, and Simulation (IMMoS) is described. The primary goals of IMMoS are to (1) improve process guidance for system dynamics modeling, (2) support system dynamics goal definition, (3) integrate dynamic models and existing static software engineering models to enhance learning, and (4) integrate methods of system dynamics modeling, static process modeling and GQM.

IMMoS combines static modeling approaches with system dynamics, extends process improvement approaches, and supports decision making. The reader is encouraged to read [Pfahl, Ruhe 2005] for detailed examples.

These integrated approaches are showing promise. However, more work is needed to define and implement methods that better combine empirical measurement and process simulation. Further experiments and case studies are necessary to validate the approaches. This will take an alignment of diverse efforts for experimentation and process simulation modeling across the software community.

2.11.3.1 Example: GQM Applied to Inspection Modeling

The process of choosing model aggregation levels and model elements is supported by the GQM paradigm. Modeling goals drive the structure and detail of models. For example, a simplified GQM comparison of the differences between two major models for inspections ([Madachy 1994b] and [Tvedt 1995]) are highlighted in Table 2-7. The different models reflect their respective research goals.

Their different goals result in different levels of inspection process detail. The purpose of the Madachy inspection model is a top-level perspective regarding when and how frequently inspections should be used. The Tvedt model focuses on optimizing the inspection process itself. In the former, the overall project effects of incorporating inspections are investigated, resulting in a more aggregated model than the latter's focus. This example will be elaborated with more detail in Chapter 3.

2.12 PROJECT MANAGEMENT CONSIDERATIONS

A modeling and simulation effort should be managed as any other software project. The average model development is relatively small compared to most software pro-

Table 2-7. Illustration of GQM applied to software inspection modeling

Model	Goal	Questions	Metrics (Model-realized Level Instances)
Madachy 1994b	Improve overall software process performance	What are the dynamic project effects of incorporating inspections?	Levels before and after inspection
Tvedt 1995	Improve the inspection process	What are optimal inspection parameter values?	Levels for each inspection activity

jects, but that does not subvert the need for best practices. Focusing on people, addressing risk, and overall diligence should be practiced nonetheless.

2.12.1 Modeling Communication and Team Issues

Communication is vitally important in modeling just like it is on software development projects. Always seek involvement from the front end per the WinWin spiral, or modeling efforts will not succeed. Make sure to identify the stakeholders and their win conditions, and consider them from the outset. Developing buy-in to the model is usually more important than the model itself. Otherwise, the results will never be used.

Modelers should be constantly talking to the organizational teams that both support and underwrite the modeling, as well as those being modeled. They need to know what is being done, their expectations might need to be managed, and they will cooperate and be much more proactive when they realize how they will gain from the process modeling. Always keep in mind that modeling in industry must be very applied and have demonstrable value. Oftentimes, a business case for the modeling should be developed beforehand.

Everyone comes with his or her own preconceived mental models and success criteria. Open communication will go far to elucidate those models and identify the clashes between stakeholders' visions of success. Eliminating the clashes between peoples' internal models is a major premise of the MBASE philosophy.

Be aware of the different perspectives of your audience. Typically, executives and those being exposed to system dynamics for the first time do not need or want to see a fully elaborated model. The model could easily alienate if they are not familiar with the notations. Always use simple diagrams to communicate with others until they seek more detail. Sometimes, a causal diagram is useful for high-level explanations. Communicate structure with simplified system diagrams, loop diagrams, and submodels. Try to facilitate interaction with the model. This will go a long way toward getting support and having the model used by others. It is also an important part of the modeling process to have experts validate the model results.

The form of the model itself can be crucial in communication. When possible, a control panel interface is useful for others to be able to view key inputs and outputs. Cluster the controls into related groups. Try to use pictures to illustrate and delineate model sections. Sometimes, a hierarchy of pictures helps to explain the relationships between model sectors.

All models contain hidden assumptions. It is, therefore, critical to present and document assumptions in a model to eliminate confusion and force clarity. Results should be qualified relative to the assumptions built into the model. Equations should have comments describing the assumptions and rationale used in their formulation.

Simple causal loop diagrams (CLDs) are sometimes better for describing a model than a fully elaborated, executable version. This is particularly important when key stakeholders do not have the skillset and/or time to comprehend model details. Often a causal loop diagram that shows the basic feedback loops of a system will get the important points across.

Documentation is an important part of communication, particularly when it is the primary form of imparting information to a group. Not everyone can attend briefings, and many times reading is necessary to understand low-level details. One should clearly describe the problem, study objectives, assumptions, simulation activities, and describe the final recommendations and their organizational impacts. See Section 2.12.3 for more details on documenting modeling projects.

As Barry Boehm said, "The models are just there to help, not to make your management decisions for you" [Boehm 1981]. A model alone is not going to make a decision. Remember that a model is not reality. It is only an aid to be used with other resources before committing to a decision. A team of people should use a model in this proper context. If the model indicates an action that totally flies in the face of realism and practicality, then you must listen to the collective intuition of the team.

Model building, understanding, and application should be a team effort for maximum leverage. Personnel who are not schooled in analytical methods and modeling approaches nevertheless should be effective participants in modeling and simulation. There is a gap between analytical specialists familiar with modeling techniques and tools and others in an organization with little or no modeling background. The quality and depth of team participation should be increased; otherwise, the modeling efforts may fail to have impact. People outside of the modeling loop often feel no ownership of the models and place little meaning or importance to the models.

People should be able to construct meaning from models through active engagement with model building (not just data collection and reporting) and model interpretation. There should be time made available for consideration of, reflection on, and communication about the models. Useful meaning is not inherent in models, but must be constructed by the model's users, that is, the entire team. Issues that must be addressed include collaboration between disparate groups, development of shared understanding, and usability of tools and approaches.

In order to address these issues at Litton, we would begin modeling efforts with a kick-off meeting for everybody with a stake in the models. This includes anyone who provides data or insight, those whose working processes are being modeled, other process "owners" as appropriate, those who need to understand modeling results for planning or management concerns, and the modelers and their sponsors (typically executives funding the effort). Brainstorming and data interpretation sessions are scheduled with those who can contribute, and all stakeholders are kept apprised of the ongoing results. It is vitally important that the model insights be demonstrated and resulting process changes discussed by those impacted.

2.12.2 Risk Management of Modeling Projects

An organizational effort that is not bought into by or relayed to others will be a lost cause and will doom future attempts. Conversely, the possible rewards of group collaboration and modeling are tremendous for process improvement. Table 2-8 shows some common modeling risks and actions that can be taken.

Table 2-8. Modeling risks and mitigation techniques

Risk	Mitigation Techniques
Not involving the right people to achieve buy-in	Work proactively with all stakeholders across organization: executive management, modeling project sponsors, process owners and champions, and other identified stakeholders.
Constantly evolving processes make the model irrelevant in the future	Parameterize the model so its subparts and parameters can be easily updated. Develop and implement a process for periodic model evaluation and updates. Train personnel.
Overly complex model attempting to capture too much or, conversely, an overly simplistic model not capturing important attributes	Balance user friendliness with model prediction figures of merit. Use surveys, workshops, and other user feedback and interface measures to gauge usability. Quantify predictability measures for different model options.
Not having right resources	Renegotiate goals and resources to be compatible.

2.12.3 Modeling Documentation and Presentation

Documenting a simulation study is crucial if others are to understand and apply the results. A model should be made as accessible as possible. Documentation is particularly important to communicate to others who were not directly involved in the study. A report should address the following items at a minimum: clearly describe the real-world problem, identify the objectives of the study, identify all assumptions, describe the simulation process activities, provide details on the resulting model and its testing, and include conclusions and final recommendations for the user/reader. If applicable, transition to other users and model maintenance should also be considered in the report.

All modeling projects and homework exercises, regardless of size, should keep the following in mind:

- Always show equations
- Liberally include comments with equations
- Identify assumptions used
- Provide rationale for numeric values and calibrations
- Show outputs for different simulation cases
- Discuss the results

Early project briefings or reviews should address the following items:

- Describe problem and background
  What will result from your study
- Show system boundary

- How will a user interact?
  Inputs and outputs
  A prototype of input and output is very useful
- Reference behaviors and other methods of verification
- Other material as appropriate (see sample report in Table 2-9)

A good report will be clear, complete, and demonstrate an organized approach to model building. The described model should also be complete and validated, or at least explain the extent of validation. It is critical that all modeling assumptions be made clear, with appropriate rationale for deriving constants and parameter values. The report will thoroughly explain sensitivity analysis and controlled experimentation to verify the model. It will include discussions of model expectations, lessons learned, conclusions, and potential improvements.

Table 2-9 provides a sample report outline that can and should be tailored for particular situations. The size of the report depends on the scope and nature of the study.

2.12.4 Modeling Work Breakdown Structure

The process steps contained in this chapter may serve as the basis for a work breakdown structure for a modeling project. The granularity of activities should depend on the nature of the study, where effort is focused, and the number of people involved. Alternatively, the SEI came up with a preliminary work breakdown structure with two major divisions: managing the modeling work and modeling the process.

Table 2-10 shows an activity decomposition elaborated from unpublished SEI work on a process guide for descriptive modeling. There is a large focus on organizational issues with little detail of model construction activities, and it should be modified for specific situations.

2.13 MODELING TOOLS

Modeling with system dynamics requires a computer simulation tool. It is highly recommended that you use an existing, proven simulation tool rather than creating your own. It will be more flexible, stable, robust, understandable, and save much time. You gain the leverage of high-level commands and an overall modeling framework.

Since the architecture for a simulation software package probably will not change during the usage of a specific model, worries about fragile demonstration prototypes are minimized. Small prototype models can be used as the foundations for enhanced, complex models without the overall architecture breaking (though the model formulation may need revamping).

It is worthwhile to mention specific commercial tools. The history of the field is linked to evolving computing paradigms as reflected in the toolsets. Dynamo was the pioneering system dynamics tool and has been around for decades [Forrester 1961], [Richardson, Pugh 1981]. It is still based on writing source equations and lacks a modern graphical user interface, but many classical references are based on it.

Table 2-9. Sample simulation report outline

Simulation Report

1. Introduction
   - Problem statement and description—extent and history of the problem, causes of the problem; possible solutions and their obstacles
   - Purpose of study
   - Purpose of model building
   - Executive summary of key results
2. Background
   - System description—include diagrams that illustrate the system configuration
   - System reference behavior—narrative description, tabular data, graphs
   - Assumptions and the underlying rationale for each
3. Model Development
   - Modeling process—include modeling approach and sequence of events
   - Model evolution
   - Data acquisition—source of data, method of collection, problems, solutions, analysis, etc.
4. Model Description
   - Timeframe, spatial boundaries, entities, attributes, key assumptions
   - Process flow and model structure—flowcharts, model diagrams, etc.
   - Key logic and equations, sources of uncertainty, probability distributions, etc.
   - Model overview—include related flowcharts, flow networks, block diagrams, etc.
   - Assumptions and other model details
   - Approach for verification and validation
5. Model Verification and Validation
   - Testing results, extreme values, sensitivity analysis
   - Comparison to reference behavior, expert review, etc.
   - Statistical analyses
   - Other testing and V&V methods
6. Model Application and Transition
   - Experimentation and analysis of results—include tabulated data, plots, bar graphs, histograms, pie charts, statistical analyses and other reports, etc.
   - Interpretation of model results
   - Limitations of the model and future enhancements
   - Next steps
   - Model transfer issues
7. Conclusions and Recommendations
   - About the real world system—including policy suggestions
   - About the model
   - About the modeling process
   - About the modeling tool(s) and platform used
   - Process improvement scenarios
   - Future research
8. Appendices (if necessary)
   - Supplementary model details
   - Model run output
   - Selected computer outputs, related letters, technical articles, etc.
   - Additional data analysis as needed

Table 2-10. Modeling activity work breakdown structure

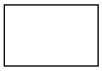
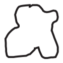
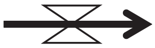
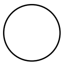
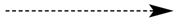
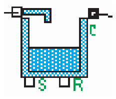
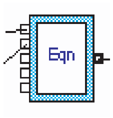
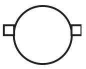
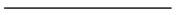
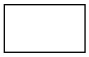
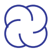
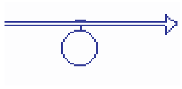
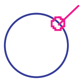
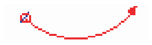
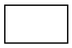
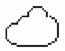
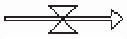
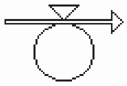
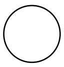
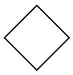
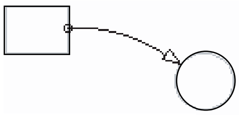
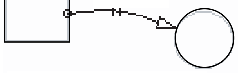
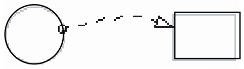
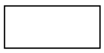
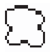
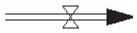
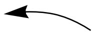
Manage the Modeling Work	
Plan Plan the product Plan the project	Establish the objectives of the modeling product and establish commitment to a specific plan for developing that product.
Develop the team	Train personnel to be able to participate in the modeling process as a cohesive group.
Contract with management	Ensure management support for the descriptive modeling process. In addition, project issues and their resolutions are identified.
Model the Process	
Conduct process familiarization Gather initial data Construct initial model	Establish a modeling frame of reference by translating existing process documentation to an initial model.
Collect data Prepare for interviews Conduct interviews Gather other data Analyze data	Gathering and assembling interview and other data for use in building the process model.
Construct model Build model Verify model	Produce and accurate and appropriate representation of the collected interview data in a process model.
Review model	Allow process experts to review and validate the process model.
Documentation	Produce documentation report and presentations.

Current popular toolsets include Extend [Imagine 2006], iThink/Stella [isee 2006], Powersim [Powersim 2006] and Vensim [Ventana 2006]. Collectively, all of them are represented in the work described in this book. These tools are visual applications whereby models can be constructed by drawing them. Some also provide additional programming utilities or access to source code for custom development.

Modeling tools are continuously improving and adapting to new technologies. One of the important new directions is for distributed and networked simulations on the Internet or an intranet. Usage of the Internet has spawned a variety of distributed and collaborative simulation features. Most of the tool vendors are still improving their networking capabilities. Another improvement is for model conversion between tools. Utilities exist for transforming a model created with one tool into a format to be used in a different modeling tool.

Notations used for system dynamics/systems thinking tools are shown in Table 2-11 for the most popular toolsets. The table can be used to help understand system dynamics diagrams generated by unfamiliar tools. A user should consider price, documentation, training, support and maintenance, computer platform, and user familiarity

Table 2-11. Tool notations

	Element				
Implementation	Level	Source/Sink	Rate	Auxiliary	Information Link
Traditional manual drawing (e.g. for Dynamo[1])					
Extend[2]					
iThink/Stella[3]					
Powersim	 Level	 Cloud	 Flow  Flow-with-rate	 Auxiliary  Constant	 Information link Cloud  Delayed info-link  Initialization link
Vensim				Variable name (no symbol)	

[1]There is no tool support for diagrams in Dynamo, hence, they are usually drawn manually using traditional system dynamics notation. Newer versions may include limited diagramming capabilities.
[2]Extend allows the substitution of custom graphics for model elements.
[3]iThink diagrams are used throughout this text. Stella is the same product for academic environments.

before committing to a specific tool. Further details on tools and their vendors are provided in Appendix B.

2.14 MAJOR REFERENCES

[Forrester 1968] Forrester J. W., *Principles of Systems.* Cambridge, MA: MIT Press, 1968.

[Forrester, Senge 1980] Forrester J. W. and Senge P., "Tests for building confidence in system dynamics models," in A. Legasto et al. (Eds.), *TIMS Studies in the Management Sciences (System Dynamics),* The Netherlands: North-Holland, 1980, pp. 209–228.

[Rechtin 1991] Rechtin E., *Systems Architecting,* Englewood Cliffs, NJ: Prentice-Hall, 1991.

[Richardson, Pugh 1981] Richardson G. P. and Pugh A., *Introduction to System Dynamics Modeling with DYNAMO,* Cambridge, MA: MIT Press, 1981.

[Richmond et al. 1990] Richmond B. and others, *Ithink User's Guide and Technical Documentation,* High Performance Systems Inc., Hanover, NH, 1990.

[Sterman 2000] Sterman J., *Business Dynamics: Systems Thinking and Modeling for a Complex World,* New York: Irwin McGraw-Hill, 2000.

2.15 CHAPTER 2 SUMMARY

Creating simulation models to understand process dynamics has much in common with other types of software development. The approach described uses system dynamics as a rich, elegant technique for modeling complex dynamic systems. A main premise of the technique is that the behavior of a system over time is a result of its own structure. Another fundamental is that system elements interact through feedback loops, where a change in one variable affects other variables over time, which in turn affects the original variable. When modeling and understanding these effects, we gain leverage to improve processes by being better informed about the dynamic results of our decisions.

System dynamics treats process entities as aggregated flows over time. This assumption greatly simplifies system modeling and makes it easier to handle interconnected factors. However, discrete-event modeling and hybrid approaches also have a place in software process modeling. There are various trade-offs between the methods and system dynamics is generally better suited for macro-level studies.

General system behaviors observed in many types of processes include goal seeking behavior, exponential growth, S-shaped growth, oscillating behavior, and combined behaviors. Information smoothing may also modulate the perception of behavior (i.e., a system actor cannot always observe true, real-time conditions). Common structures are available in system dynamics that mirror real processes and cause these observed behaviors.

A top-down iterative approach to modeling will make it easier in the long run. A risk-driven approach using the WinWin Spiral life cycle works well to help structure the modeling activities and mitigate risks. One cycles through problem definition, model conceptualization, formulation, simulation, assessment, and policy analysis. Once again, the purpose is to better understand decision consequences that drives the scope of study and the subsequent modeling process.

Problem definition sets the stage for modeling. You explicitly define a purpose that addresses organizational goals and is important to other stakeholders. The degree of implementation is also addressed in terms of model form and detail, rigor used, and institutionalization of new process policies. Reference behaviors, which are plots of key variables over time, are generated during problem definition. These behaviors help to gel ideas and are used later during validation of the resulting model.

During model conceptualization the system boundary is defined and a top-level view of system elements is created. Be extremely careful in selecting the portion of reality to model so that it does not include too much or too little. Then you identify physical structures and information flows, distinguish perceived from actual information, and identify decisions made by system actors. Causal loop diagrams can be handy at this stage to portray cause and effect relationships and feedback loops. In some studies, this might be all that is necessary in order to increase understanding.

In model formulation and construction, the high-level concepts are elaborated into detailed equations. The system is described in terms of levels, rates, auxiliaries, and feedback loops. Levels (stocks) represent accumulations in a system. They can change the shape of an input over time and potentially decouple inflow and outflow rates to allow independence, and, as a result, can also induce disequilibrium behavior. A single-level system cannot oscillate, but a system with at least two levels can oscillate because one part of the system can be in disequilibrium. Choosing to model a concept as a level or not depends on if it can be perceived as a stock or accumulation over time. The snapshot test can be handy for determining this.

Rates are the flows that occur in conjunction with levels. They are not instantaneously measurable, and can only be averaged over time. No rate can control another rate without an intervening level variable. Rates may have constant, variable, additive, and/or multiplicative formulations. Rates are essential components in delays, positive feedback loops, and negative feedback loops. System information is captured in auxiliary variables to help elaborate the stock and flow structures, such as containing score-keeping information for rate decisions.

Time delays are found everywhere in processes. Common examples include first-order delays, third-order delays, and pipeline delays. Delayed outflows account for the average time that entities remain in a level and can model artifact transformations or assimilation processes. Negative feedback exhibits goal-seeking behavior and may cause system instability. Positive feedback produces a reinforcing growth or decline process.

During simulation and assessment, a model is verified to be free of defects and validated by running it under different conditions. Many aspects of structure and behavior are evaluated with a wide range of tests. First, the model is put into steady state, test inputs are used, parameter sensitivity, structural sensitivity, and robustness tests are run. You assess whether the model is suitable for its expressed purpose, its consistency with reality, and its overall utility and effectiveness for the end user.

A policy represents a decision on how processes should be performed. During policy analysis, we seek to understand the real-world impact of process changes. Similar to the wide range of tests during model validation, we experiment with changes to policy parameters and structures to assess policy validity, robustness, suitability, and feasibility. Familiarity and insight into real-world problems is critical to judge policies. Human value judgments must be made in order to understand policy trade-offs and determine the most desirable policies.

In the spirit of continuous process improvement, models should be continually rechallenged, assessed for relevancy, and refined. Upon further inspection, model shortcomings may come to light that should be addressed. It is also possible that the goals, constraints, and objectives of the organization may have changed as well as the process environment being modeled.

Quantitative modeling depends on having data, but the lack of good data should not hold up modeling efforts. Techniques are available to cope with missing data. GQM is a valuable framework for many aspects of data collection and modeling. By identifying goals and deriving relevant questions and metrics, GQM ensures that whatever you do can be mapped back to organizational goals and that you are focusing on important metrics data.

Project management of modeling and simulation projects is an important consideration. Concentrating on people, communication, and risk management are some best practices to keep in mind. Documentation is important just like for other software development and evolution projects.

2.15.1 Summary of Modeling Heuristics

Most of the following heuristics were discussed in greater detail in preceding sections. They have been identified for system dynamics modeling, but almost all of them also apply to other modeling methods. As with all guidelines, these generally apply in most situations but the modeler must always interpret them in the context at hand. Some may even conflict with each other in given situations.

Note that these heuristics do not repeat the essential mechanical steps such as "develop a reference behavior," "generate test inputs," "do sensitivity analyses," and so on. Rather, these are guidelines that help in structuring a solution.

General Modeling

No model is perfect.

All models are incomplete.

A model is not reality.

It is possible to build many different models of a single process.

All models contain hidden assumptions.

Continually challenge the model.

The models are just there to help, not to make your management decisions for you.

Problem Identification

A model is created to answer specific questions.

Consider the audience, desired policy set, and level of implementation when defining the model purpose.

Define the problem dynamically in terms of reference behaviors, even if hard data is missing.

Strive for relative measures in the reference behaviors.

Model Conceptualization

Define a clear, operational purpose of the model.

Within the system boundary, include the parts necessary to generate the behavior of interest.

Do not try to model the "system."

Aggregate and abstract to the appropriate degree.

Use a top-down iterative approach.

KISS (keep it simple, stupid).

Model Formulation and Development

Do not enumerate all factors at first.

Iteratively refine and slowly add relationships to model.
Normalize when possible.
Use relative measures.
Do not stray too far from a simulatable model.
Do not model in isolation; try to involve those being modeled.

Model Validation

Look for qualitative similarity on the first pass.
Alter one parameter at a time at first.
Be conscious of reality constraints.
Model validity is a relative matter.

Data Collection

Model design should not be postponed until all pertinent parameters have been accurately measured.

Communication

Use simple diagrams to communicate with others until they seek more detail.

2.16 EXERCISES

2.1. What are the model implications, in terms of sinks and sources, as model boundaries change when focusing on different software engineering activities such as requirements, design, coding, and testing?

2.2. What are the model implications, in terms of sinks and sources, as model boundaries change for progressively larger organizational entities such as project, project portfolio, department, region, and organization? You can choose your own environment or that of another specific organization.

2.3. Provide more specific examples of the general system behaviors observed in software or systems processes.

2.4. Create your own software process domain examples of a first-order delay, negative feedback, and positive feedback.

2.5. Create your own examples of each of the basic rate equation patterns in the software process domain.

2.6. Create a simple example system that demonstrates a coincident flow ("coflow").

2.7. Create a graph function. Choose an important relationship between software process factors that can be modeled with relative scales. Describe the rationale behind the endpoints and slopes.

2.8. Explain why and in what situations you would use a graph function instead of an analytic equation.

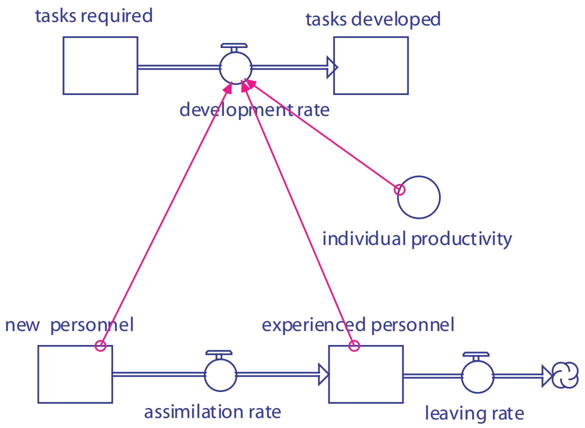
2.9. What effects does the smoothing time interval have on information smoothing?

The following three exercises should be done manually without the use of computer simulation. A graph pad is handy for sketching the curves.

2.10. (Do these manually, without a computer.) Given the system diagram below and the following:

- Simulation time is measured in months
- New personnel INIT = 60
- Experienced personnel INIT= 10

a) Identify the units for all model elements.



Draw the levels of new and experienced personnel from 0–10 months for the following cases. Clearly label the axes and put numbers on them, so the quantities over time can be easily read.

b) assimilation rate = 5

leaving rate = 0

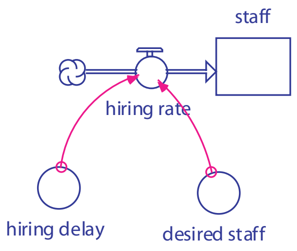
c) assimilation rate = 6

leaving rate = 2

d) assimilation rate = pulse of 10 at time = 3

leaving rate = 0 for 0 < time < 5, and = 4 for 5 < time < 10

2.11. (Do this manually, without a computer.) Below is a simple hiring model, where the simulation time is measured in months. Identify the units of measurement for each entity in the model. Also sketch the output for staff level for three cases: (1) hiring delay = 0.5, (2) hiring delay = 1, and (3) hiring delay = 3. Your sketches do not have to be exact, but the three cases should be clearly delineated on the chart as best you can draw them. Assign numbers to the y-axis for the plots.

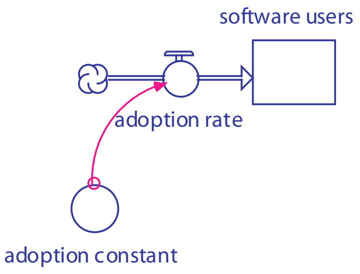


```
▭ staff(t) = staff(t – dt) + (hiring_rate) * dt
    INIT staff = 0
    INFLOWS:
      hiring_rate = (desired_staff-staff)hiring_delay
○ desired_staff = 1.0
○ hiring_delay = 1
```

Fill in the blank below:

This model is an example of ____________________ feedback.

2.12. (Do this manually, without a computer.) Below is a software adoption model, in which the simulation time is measured in months. Identify the units of measurement for each entity in the model. Then qualitatively sketch the plot of software users.



```
software_users(t) = software_users(t – dt) + (adoption_rate) * dt
INIT software_users = 10
INFLOWS:
    adoption_rate = adoption_constant*software_users
adoption_consant = 1.2
```

Fill in the blank below:

This model is an example of ____________________ feedback.

2.13. a) How would you test a model to determine how sensitive it is to parameters whose true values are unknown?

b) Identify three aspects of model structure and behavior that should be resolved before such tests are performed.

2.14. Define and demonstrate the differences between exponential growth and goal-oriented growth in terms of rate equations and output graphs.

2.15. Identify a particular problem in your environment and draw causal loop diagrams of the major effects.

2.16. Choose an ongoing software process and perform the snapshot test. Document your conclusions as to what are stocks and flows.

2.17. Explain the importance of test functions in the construction of dynamic models.

2.18. Describe and give examples of three types of model sensitivity analyses.

2.19. In a well-designed model, how do you treat variables that change more slowly or change much more rapidly then the variables of primary interest in the system?

2.20. If you have never run a simulation, choose some of the provided models and begin using them to get familiar with simulation operations.

2.21. If you have never modified a simulation model, choose one or more of the provided models and make some simple changes. Rerun them to see the effect of your changes.

2.22. Create a set of concise models that produce the different general behaviors. Parameterize them so the output shapes can be experimented with.

2.23. Create a set of concise models that implement the different rate patterns, delays, and negative and positive feedback.

2.24. Undertake some of the Brooks's Law model improvements noted in the text (beware of partitioning because of its difficulties), or improve some other aspect of it. Also see the Brooks's Law model exercise in Chapter 6.

2.25. Choose any of the software process feedback examples in [Weinberg 1992] and draw causal loop diagrams to represent them.

Advanced Exercises

2.26. Elaborate the causal loop diagrams created from [Weinberg 1992] in the previous exercise into simulation models. Run experiments and document the lessons learned.

2.27. Create a simple model of your environment to evaluate the attainment of a measurable corporate goal. Does your output resemble any of the generic behaviors? Also perform sensitivity analysis by varying the process parameters. How do the different inputs affect the goal-seeking behavior curve?

2.28. Identify some policies to study in your environment that will form the basis for a simulation term project. Start the report using the documentation outline.

2.29. Identify a software process topic that will form the basis for a simulation term project. Start the report using the documentation outline.

2.30. Identify a research problem and apply GQM to determine what relevant measurements a process model should include.

2.31. Identify a software process research problem and find or develop reference behaviors for it.

2.32. Identify a major research problem that you will address beyond the scope of a course term project. Start writing down the description using the provided documentation outline or your own format.

2.33. Choose a policy addressed in one of the provided elaborate models. Use the model to go through the steps of policy analysis. Document your process and conclusions.

2.34. Model a feedback process for implementing new policies.

2.35. Choose one of the provided small-scale models and put it into steady state.

2.36. Study the Abdel-Hamid project model (or one of the other provided elaborate models), describe how to put it into steady state, and suggest validation tests.

2.37. Draw a rough reference behavior for one or more of the following trends for a given environment. Do not refer to any source except your own insight. Qualify the context of your graph. Is the behavior for the last, current, or a hypothetical project? Does it represent an organizational trend for multiple projects?

- Requirements changes over time
- Estimated size over time
- Staffing profile over time
- Defect fixing cost over time
- Defect density over time

- Productivity over time
- Schedule performance index (SPI)
- Cost performance index (CPI)

The following refer to multiproject organizational trends:

- % reuse over time
- % COTS over time
- Defect detection efficiency over time
- Estimation quality factor over time

2.38. In a sort of reverse Brooks's Law, model the effect of extra work required when someone leaves a project. There is often additional coordination work needed to sort out the lost person's tasks. Since there are fewer workers and the project has slowed down, management may increase the pressure. This may then lead to others leaving.

2.39. Choose one or more of the following general process/product measures, perform the necessary research and analysis to assign them reasonable numeric values to use in a model (or for the purpose of evaluating a model that calculates them). Either use average values, define a reasonable range, or develop a factor relationship (e.g., average productivity as a function of size). Consider the metrics within the context of a particular organization or a defined meta-group (e.g. a country or type of company), and make them process specific. For example, productivity can be defined as overall productivity, design productivity, testing productivity, and so on.

Qualify your values, describe your analysis, and optionally show the background data used. Is the measurement context for your last, current, or next project? An organizational average? For an upcoming model of your process?

- Productivity
- Defect density
- Defect detection efficiency (yield)
- Defect finding and fixing effort
- % of effort on rework
- Assimilation delay
- Hiring delay
- Learning curve rate
- % effort on requirements
- % effort on design
- % effort on programming
- % effort on testing
- % effort on system engineering
- % effort on quality assurance
- % effort on peer reviews
- % effort on software process improvement
- % effort on transition activities
- Cost drivers from COCOMO II or other software cost model
- Defect prediction factors from a defect model
- % requirements evolution and volatility
- % reuse
- % COTS
- % modification
- Financial overhead percentage
- Other measures relevant to your concerns

2.40. Go through each of the validation tests and assess the Brooks's Law model. Create a validation report using the test matrix as an outline. You will need to create and run your own tests.

2.41. Begin a study to compare different software process life cycles, such as the iterative versus the waterfall. Chapter 1 provides a nominal progress reference behavior for that comparison. Other possibilities include a comparison of agile methods, transformational approaches (e.g., fourth-generation languages), or any defined life cycle being used. Examine the different processes in terms of what accumulates and flows, and identify structural differences in their main chains. When devising a comparison model, strive to keep all project factors equal except for the life-cycle process. It is also suggested to compare only two life cycles at a time to simplify the project. Branch out into more life cycles after initial success with comparing two.

2.42. Take one or more of the modeling risks and expand on their description(s) and specifics as to how the mitigation activities might work or not. Or identify your own modeling risk(s) and develop detailed mitigation plans.

2.43. Identify more modeling heuristics; explain their usage and significance.

2.44. Take a modeling heuristic and study it in more detail. Examine and write up its nuances. Provide detailed rationale for specific situations in which it is applicable, or counterconditions in which it may not be. Modify the heuristic accordingly for different contexts.