4

PEOPLE APPLICATIONS

4.1 INTRODUCTION

People offer the highest leverage opportunity to improve process performance and increase the chance of project success. Software processes will always be sociotechnical by nature, and due attention should be paid to the human aspects. Unfortunately, people issues have been less represented in process modeling and simulation compared to technical aspects. Abdel-Hamid's model had a large portion dedicated to people aspects [Abdel-Hamid, Madnick 1991] and other models have stocks for personnel staffing levels, but only a few other efforts referenced in this chapter explicitly assess people attributes and trade-offs.

Many proposed "silver bullets" for software engineering focus on technical solutions while ignoring the people dynamics. People must be available. They should have requisite skills and experience for their jobs, the ability to work well with others, and they should be adequately motivated to perform, have necessary resources, and have a good working environment that fosters creativity and learning. The best designed processes and latest technology will not help a project without human cooperation. Agile approaches recognize that processes are ultimately executed by people. No process will work without people and, therefore, "people trump process" [Cockburn, Highsmith 2001].

The focus of this applications chapter is on phenomena directly related to people. Constructs are shown for important people aspects including motivation, exhaustion, experience and learning curves, skill development, training, hiring and retention, com-

munication, stakeholder collaboration, and workforce dynamics at the project and macro levels. Even though many of the people attributes are considered "soft factors," it is feasible and meaningful to model them. One strength of system dynamics is the ability to model soft factors and easily integrate them with hard factors.

Due to the relative scarcity of people-oriented models, this chapter includes a few areas for which there are currently no example models. These are important topics and it is expected that relevant models will be created in the future.

It can be argued that virtually all process, product, and organizational phenomena are impacted by human actions to some degree. However, this chapter will overview applications and research centered on people issues. When the work also involves more than people concerns, only relevant portions will be described in this chapter, with references to the rest of the nonpeople aspects.

One view of system dynamics is that the purpose of all models is to provide understanding to people. Thus, all models can be considered training applications, but the model topics have to be compartmentalized for the sake of organization. However, there are applications purposely created for education and training and they are covered in a corresponding section. These simulation applications were explicitly devised for classroom and other teaching venues, and typically use hypothetical project scenarios for analysis and making decisions instead of actual "live" or "to-be" projects (though they could also use them). Education and training applications have enormous potential to be tapped.

How important are people characteristics relative to other factors? Our COCOMO II research in [Boehm et al. 2000] provides quantitative relationships for major software project factors. Data shows that the combined people-related factors provide for the widest range of software productivity impact against all other cost factor types (process, product, project, and organizational).

Figure 4.1 shows the linear range of variation for software productivity due to factors directly related to people. For example, the range of 1.76 for *Programmer Capability* indicates that productivity may vary by 176% between the lowest and highest ratings. Note that the factor for *Team Cohesion* covers stakeholder negotiation and collaboration between people (an application discussed in this chapter). The total productivity variation due to direct people factors in the figure is 2008%, including 302% for the combined experience factors and 353% variation due to capability factors. The entire set of factors and their productivity ranges are listed in Appendix E and further described in [Boehm et al. 2000].

Keep in mind that the COCOMO II multipliers are from a macro, static model essentially based on regression techniques. The challenge of dynamic modeling is to move away from a static representation and consider the underlying mechanics of how these phenomena work together and the associated nonlinearities. Additionally, the COCOMO II data is from mature organizations from which data is collected; good management is assumed and, likewise, good motivation. Unfortunately, these are not always the case so nonlinear, fringe conditions also need to be considered. A more holistic view of dynamic interactions will enable better insight and decision making.

Communication overhead is another important factor due to people, but is not included in these calculations because it is represented in the size penalty in the

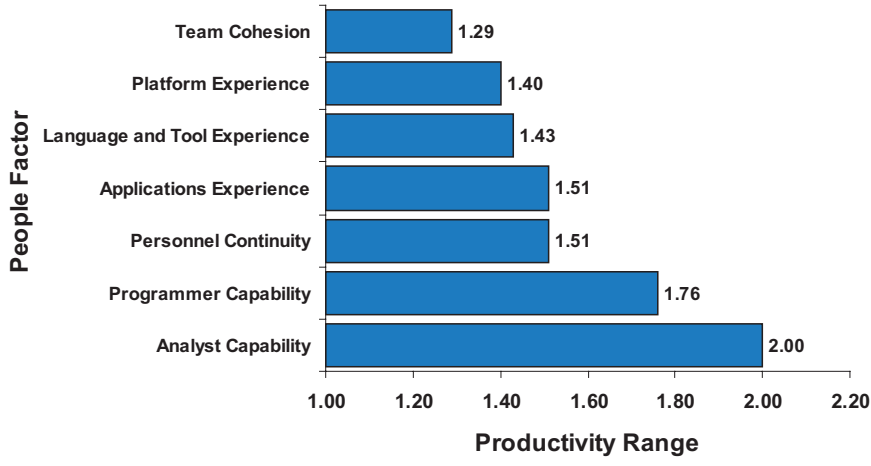

Figure 4.1. People factor impacts from COCOMO II. What causes these?

COCOMO model. Other scale factors in COCOMO II attributable to people in lesser degrees include *Architecture/Risk Resolution,* which depends on top architect availability, and *Precedentedness,* which has an experience component. Many of the other COCOMO factors are also influenced by human actions, so the overall variation due to people is actually greater than 2008%. Other static cost models include additional people factors such as management experience and capability, office ergonomics, team assignment, fragmentation, intensity, and more (see [Cost Xpert 2003], for example).

However, COCOMO assumes that people are reasonably well motivated (a corollary of the assumption that projects are reasonably well managed) and that other soft factors can be treated as invariant across projects. These assumptions do not hold true in many instances, so some people factors not currently represented in COCOMO will be addressed in this chapter. Most of these represent measures that are harder to define and collect in the COCOMO context, so some means of quantification will be shown.

An opportunity tree for people-related improvements is shown in Figure 4.2. The tree is based on our USC course notes for software engineering economics in [USC 2004] and includes some enhancements from [Bhatnagar 2004]. The tree amply shows the necessity of organizational commitment to employee growth and development. These people strategies are all areas that can be explored more through modeling and simulation, and some are reviewed in this chapter.

This chapter primarily deals with past work, and there are critical people areas that have not yet been covered in the simulation literature. Thus, the reader should be aware of some important and classic references on people considerations for software processes that do not deal explicitly with modeling and simulation. *Peopleware, Productive Projects and Teams,* by Tom DeMarco and Tim Lister [Demarco, Lister 1999] is an insightful book for those wanting to understand how to improve productivity in people. We require this book as a text for our graduate software engineering courses at USC. One of highest leverage improvements that DeMarco and Lister identify is office

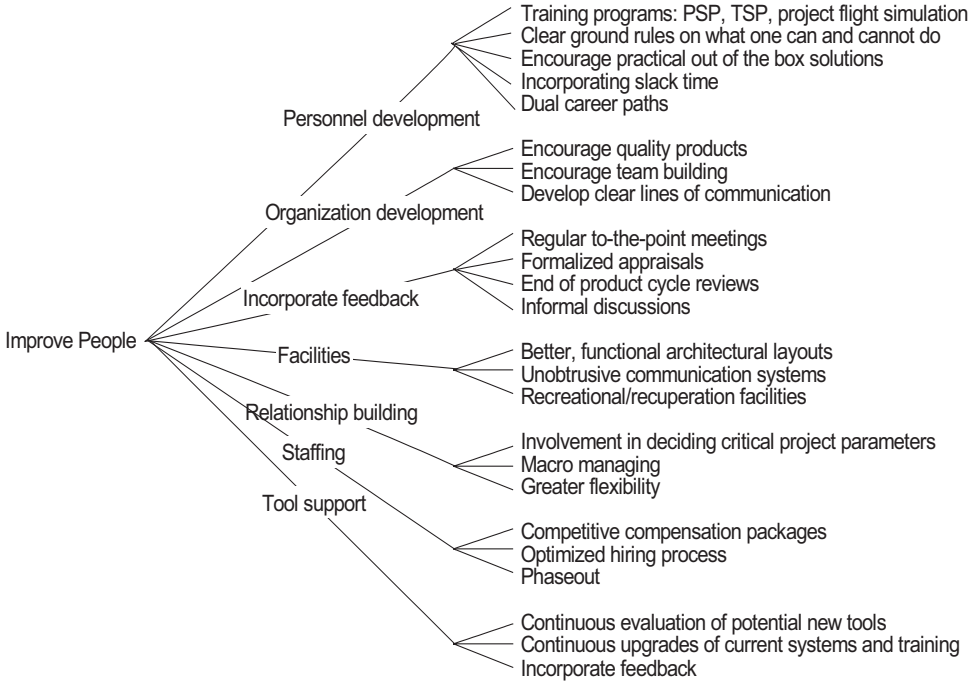

Figure 4.2. People opportunity tree.

ergonomics, but ergonomic factors have not even been modeled yet for software processes.

The 1971 classic *The Psychology of Computer Programming* by Gerry Weinberg was updated in 1998 [Weinberg 1998]. This is the first and still most comprehensive treatment of the psychological aspects of people who develop software and sheds light on what makes them tick. Weinberg describes the inner workings of programmers, but these phenomena also have not been addressed in models to date.

Another valuable reference for managers is *Managing Technical People* by Watts Humphrey [Humphrey 1997]. It is an insightful guide for leading technical software professionals using sound management principles. Humphrey gives advice on how to identify, motivate, and organize innovative people. People must be understood and respected in order to be fully dedicated. Though there are special challenges with technical people, taking the right steps can improve efficiency and quality. The resounding message is that people are the overriding important factor for software project success.

The People Capability Maturity Model [Curtis et al. 2001] complements and expands on [Humphrey 1997] by providing a structured maturity model for getting the best out of people. It also complements the software CMM and CMMI because it addresses the workforce practices for continuous improvement. This compilation outlines tangible actions that an organization can take to assess and improve their people maturity.

The handbook, *A Software Process Model Handbook for Incorporating People's Capabilities* [Acuña et al. 2005] is a valuable contribution on people-related aspects. The focus is on extending software process definitions to more explicitly address people-related considerations. It provides a capabilities-oriented software process model formalized in UML and implemented as a tool.

The guidelines in these classic books and subsequent references help to understand people dynamics and provide substantial fodder for future modeling and simulation applications. Discrete models might be suitable for some of these people effects in addition to system dynamics. The reader is encouraged to study these references and examine the chapter exercises based on them.

4.2 OVERVIEW OF APPLICATIONS

This chapter starts with a review of the detailed human resources sector in the Abdel-Hamid integrated project model from [Abdel-Hamid, Madnick 1991]. It is historically significant and covers some classic phenomena with structures for hiring, assimilation, training, and transferring of people on a project.

Exhaustion and burnout have large effects on productivity and are intimately related to motivation. These are vital people factors to be concerned about, but are rarely considered on projects. Some example models are shown, including one that is part of the Abdel-Hamid integrated model.

Learning is a broad topic that is addressed from the perspective of continuous modeling. Models to implement experience and learning curves in the context of software development are described. They are also contrasted with alternate learning curves and experience impact models.

One of the people issues for hybrid processes is how to compose the teams in terms of skill sets and responsibilities. The next section describes a model to find the optimum number of agile people to match anticipated change traffic on a very large project.

Next is a section on critical people-related areas, yet uncovered with full models. The all-important aspect of motivation is included. The effect of motivation may overwhelm any other attempt to improve process performance. Motivation is a classic case of a "soft" factor and some ways are shown to model related phenomena.

The next topics include personnel attributes such as skills, team communication, stakeholder negotiation, and collaboration factors. These application areas are described in general with few supporting models, though the references will contain some.

Understanding the dynamics of hiring and retention is necessary to maintain a ready organization. Standard modeling constructs and integrated models for assembling and maintaining teams are described. Issues of workforce dynamics at the macro level are also covered.

Simulation for personnel training is a different class of application. Experiences and research into the effectiveness of using simulation for personnel training are summarized from several studies. For more on this topic, the reader is encouraged to read the references in that section, and to look ahead to Chapter 7.

4.3 PROJECT WORKFORCE MODELING

Workforce modeling quantifies personnel resources over time. Separate levels are frequently used to represent different experience or skill pools. The number of levels used to represent different classes of people will depend on the environment and modeling goals.

4.3.1 Example: Personnel Sector Model

The personnel sector of the Abdel-Hamid model covers personnel hiring, assimilation, training, quitting, and transferring of people onto or off the project. It is summarized in Table 4.1.

Figure 4.3 shows the structure of the personnel sector from the Abdel-Hamid integrated project model. It is driven by the number of people sought with hiring and transferring controlled by personnel decision policies. The gap between *total workforce* and *workforce level sought* is used to control hiring and transfer rates considering schedule completion time, workforce stability, and training requirements.

New and experienced people are differentiated as separate levels for a couple of reasons. Productivity differences between new and experienced people are modeled this way. Another important factor is that experienced people are involved in training new hires. The training overhead on the part of the experienced people gives them less time to devote to other development tasks, and providing two levels allows the differentiation of which resources are consumed in training. Experienced people may work on multiple

Table 4.1. Personnel sector model overview

Purpose: Planning
Scope: Development Project

Inputs and Parameters	Levels	Major Relationships	Outputs
• Hiring delay • New hire transfer delay • Experienced transfer delay • Assimilation delay • Trainers per hiree • Average daily manpower per staff • Workforce level needed (from planning sector) • Maximum hirees per staff • Average employment time	• Workforce Newly hired Experienced	• Workforce level sought • Delays • Hiring constraints • Training effects	• Staffing profiles per workforce type • Workforce gap

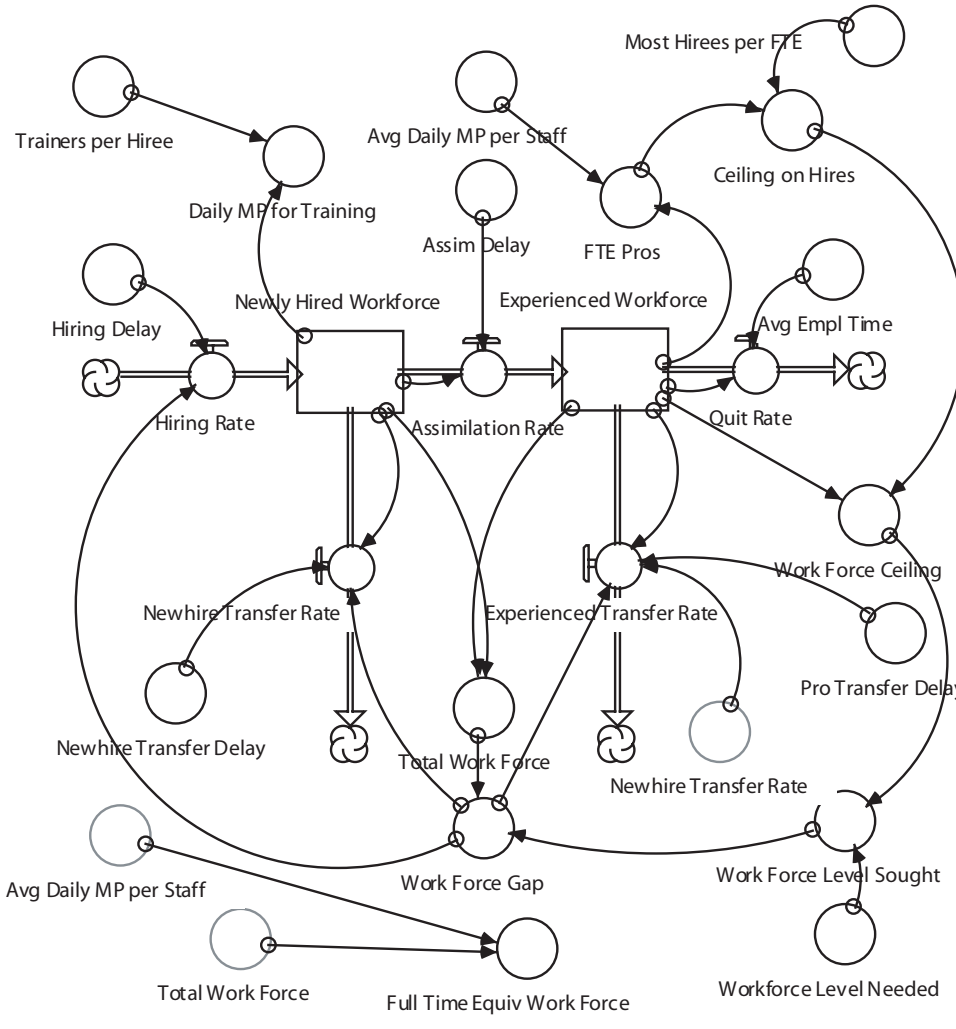

Figure 4.3. Personnel sector.

projects at once, so they may be diverted from the project in addition to training new hires. Both the assimilation rate between new hires and experienced workforce, and the quit rate representing turnover are modeled as first-order exponential delays.

Policies represented in closing the work gap via hiring and transferring decisions are important dynamic drivers of this sector. The *workforce level needed* parameter represents the decision on how many people are currently required for the project. It is calculated in the planning sector based on the current schedule completion date and perceived tasks remaining. However, the needed workforce is not always the hiring goal due to concerns for stability and the ability to take on more people. See Chapter 6

in the section on the integrated Abdel-Hamid model for the specific policy formulations in the planning sector used to make decisions based on workforce stability and the pressure to change the workforce.

Competing desires for workforce stability and schedule attainment change through the project. Towards the end of the project it would take too much time to integrate new people into the team. The number of personnel is limited partly on the ability to assimilate new people onto the project via the constraint *ceiling on new hires.*

There is also a *workforce level sought* parameter, which is used in conjunction with the *workforce level needed* parameter. The maximum number of personnel is the sum of the *ceiling on new hires* and the *ceiling on total workforce.* It is then determined whether new employees will be hired or if people are to be transferred off the project. New hires and transfers all entail delays that are modeled as first-order exponential delays.

In addition to this workforce dynamics structure, the Abdel-Hamid model also has provisions for human characteristics that impact productivity. These important factors include exhaustion and burnout, which are reviewed in the next section.

4.4 EXHAUSTION AND BURNOUT

Human nature dictates that the increased productivity effects of overtime can only be temporary, because everyone needs to "de-exhaust" at some point. This has been observed across all industries. It even held true during the Internet boom of the late 1990s in some of the hottest start-up companies. People start working a little harder with some schedule pressure, but after several weeks fatigue sets in and productivity drops dramatically. Then there is a recovery period during which people insert their own slack time until they are ready to work at a normal rate again.

4.4.1 Example: Exhaustion Model

This example models the phenomenon of exhaustion due to overwork. The representation is based on the Abdel-Hamid integrated project model. A high-level summary of the model is in Table 4.2.

The underlying assumptions of this exhaustion model are:

- Workers increase their effective hours by decreasing slack time or working overtime.
- The maximum shortage that can be handled varies.
- Workers are less willing to work hard if deadline pressures persist for a long time.
- The overwork duration threshold increases or decreases as people become more or less exhausted.
- The exhaustion level also increases with overwork.
- The multiplier for exhaustion level is 1 when people work full 8 hour days, and goes over 1 with overtime. The exhaustion increases at a greater rate in overtime mode up to the maximum tolerable exhaustion.

Table 4.2. Exhaustion model overview

Purpose: Planning, Training
Scope: Development Project

Inputs and Parameters	Levels	Major Relationships	Outputs
• Nominal fraction of daily effort for project • Exhaustion depletion delay • Maximum tolerable exhaustion • Overwork duration multiplier	• Fraction of daily effort for project • Exhaustion	• Exhaustion flow • Overwork duration threshold	• Fraction of daily effort for project • Exhaustion • Overwork threshold • Software productivity (indirectly)

- The exhaustion level slowly decreases with an exhaustion depletion delay when the threshold is reached or deadline pressures stop.
- During this time, workers do not go into overwork mode again until the exhaustion level is fully depleted.

Figure 4.4 shows the exhaustion model structure, which is a portion of the overall Abdel-Hamid project model. Figure 4.5 shows the graph function for exhaustion flow and Figure 4.6 shows the function for the multiplier to overwork duration threshold. See the equations in the provided model for a full description.

People naturally exhibit different breaking points and rest times. Table 4.3 shows typical maximum workweeks for different environments, though thresholds obviously differ by individuals. A serious workaholic may be able to put in 80-hour weeks for a few months and only need a few days rest between projects, whereas someone else might need a three-week vacation. This model used the following average values to describe an aggregate project situation:

- Exhaustion depletion delay time = 20 days
- Maximum tolerable exhaustion = 50 days
- Nominal fraction of man-days for the project = 0.6

The burnout parameters also vary by software development class (e.g., mainstream, civil service, start-up) similar to the differences in motivation. The values in this example approximate a mainstream situation. The aggregate parameters should be carefully reviewed for a given class of software development, such as the maximum tolerable exhaustion. The tolerable exhaustion is correlated to the productivity multipliers shown in Figure 4.26.

A run of the exhaustion model is shown in Figure 4.7 using these inputs and other default parameters from the Abdel-Hamid integrated project model. It clearly shows increasing exhaustion up to a breaking point. The actual fraction of man-days for the

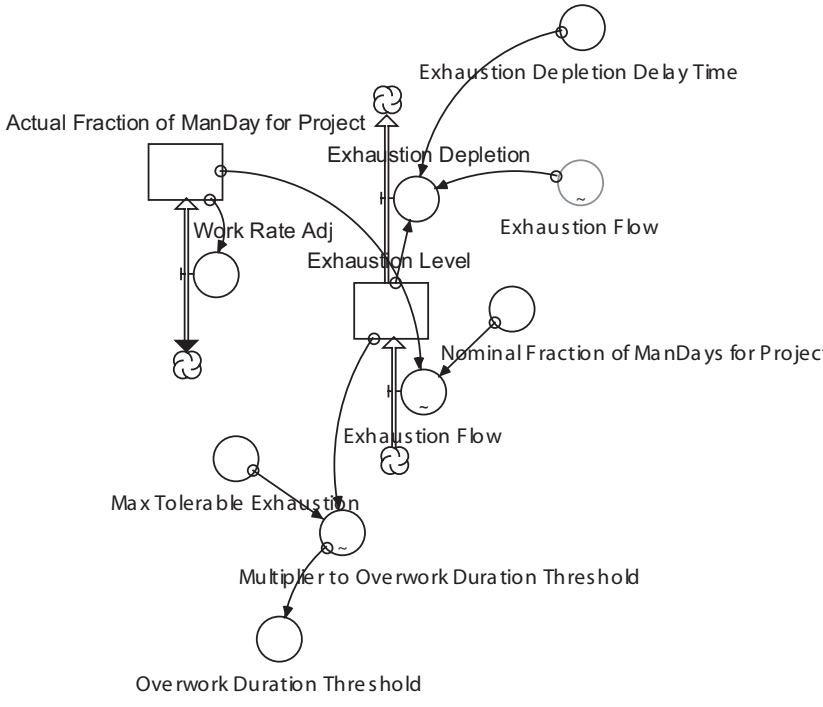

Figure 4.4. Exhaustion model structure.

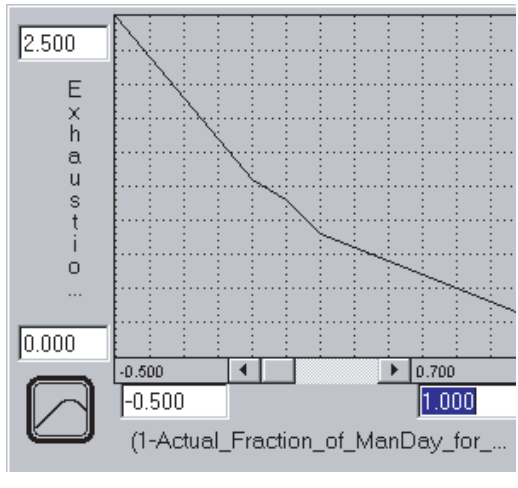

Figure 4.5. Exhaustion flow [exhaustion flow vs. (1 – actual fraction of mandays for project)/(1 – nominal fraction of mandays for project)].

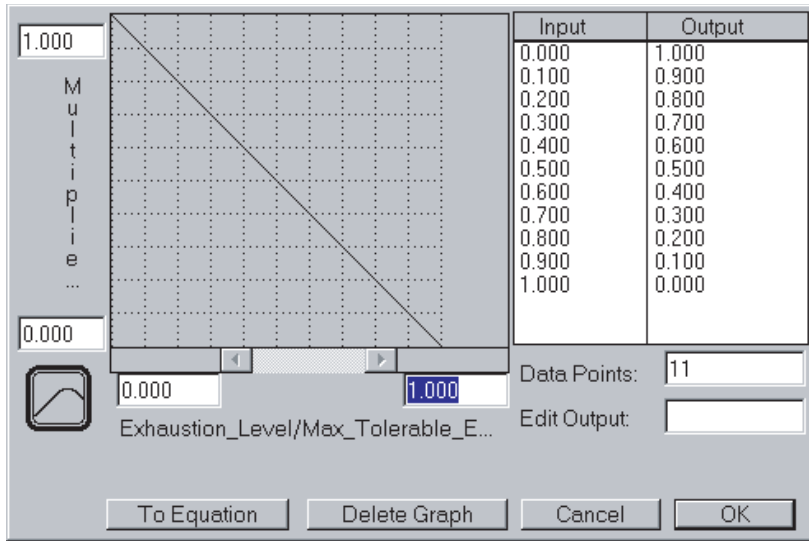

Figure 4.6. Multiplier to overwork duration threshold (multiplier to overwork duration threshold vs. exhaustion level/max tolerable exhaustion).

project rises from 0.6 to over 1 as the project falls behind schedule, which means that people have less and less slack time during their day. The fraction increases as the work rate adjustment kicks in. As the exhaustion level accumulates, it affects the overwork duration threshold such that the number of days people will work overtime starts to decrease.

These general trends continue and productivity is increased nearing the point of maximum exhaustion. When the overwork duration threshold reaches zero, the team cannot continue at the same pace, so the de-exhausting cycle starts. The actual fraction of man-days for the project and the exhaustion level both slowly decrease. The fraction of man-days decreases faster because people suddenly stop overworking. The overwork duration threshold begins to increase again, but a new overwork cycle will not start until the exhaustion level reaches zero.

The de-exhausting phenomenon is related to the concept of slack described in [DeMarco 2001] (see Section 4.7.1.2). Rest time is needed for people to gain energy and think creatively. Alternative model formulations for burnout dynamics have been created by High Performance Systems [Richmond et al. 1990] and Pugh [Richardson, Pugh 1981]. See the model *burnout.itm* as an example.

4.5 LEARNING

Learning is the act of acquiring skill or knowledge through study, instruction, or experience. In the context of a software process, developers become more productive over

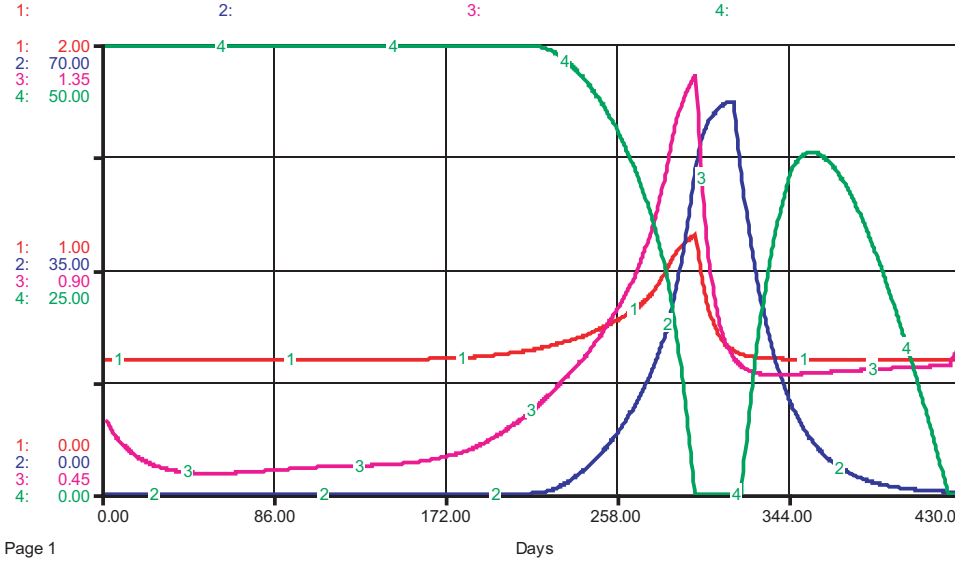

Figure 4.7. Exhaustion model behavior (1: actual fraction of man-days for project, 2: exhaustion level, 3: software development productivity, 4: overwork duration threshold).

the long term due to their accumulated experience. The increase in productivity occurs indefinitely, and a *learning curve* describes the pattern of improvement over time.

Many people are familiar with the general shape of a learning curve as expressed by a productivity function over time, such as in Figure 4.8. The curve shows an initial period of slow learning in which the development process is being learned, then a middle period of fast learning, followed by a decreasing-slope portion that nearly levels out (the form of a typical S-curve). The reduced learning portion is due to other constraints on productivity that are reached, such as machine limitations. There are always parts of

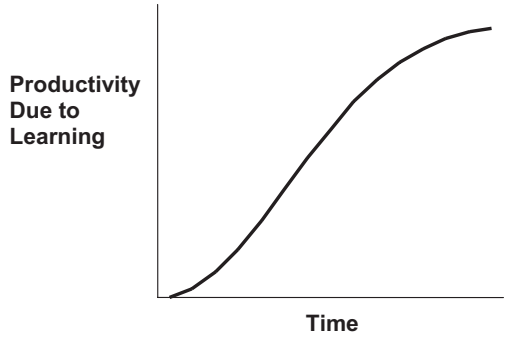

Figure 4.8. Learning as productivity over time.

the process that are constrained by machine-paced tasks such as waiting for compiles, print jobs, computer downtime, network access, and so on. Calibration of this curve for software development is treated later in this section.

The implications of learning while executing software processes can be substantial when trying to plan effort and staffing profiles. Unfortunately, few planning techniques used in software development account for learning over the duration of a project. The standard usage of COCOMO, for example, assumes static values for experience factors on a project. But learning is a very real consideration that can be easily handled with system dynamics. This section will describe techniques for expressing learning curves over time.

Learning curves are a family of equations. They have been treated in detail in many industrial applications, but have received scant attention in the software process literature. The most comprehensive treatment of learning curves in software engineering to date is an article by Raccoon [Raccoon 1996]. He ties together concepts of process stability and improvement with learning curves. For example, he points out that disruptions in the process affect learning curve offsets but not the overall slope. Some of the formulaic discussion of learning in this section is derived from his paper.

The existence of learning conflicts with some approaches to process stability. For example, a stable process is normally assumed to have a constant productivity over time. Calibration of cost models is usually performed against past project data, assuming that the calibration holds for future projects. Note, however, that some models such as COCOMO II have a provision for learning in experience cost drivers. In particular, COCOMO II has cost drivers for applications experience, platform experience, and language/toolset experience. These cost drivers provide effort multipliers for coarse ranges of experience.

Learning curves are used in several of the models throughout this book. Some of the Litton Systems case studies embody learning curves to model the effects of major process disruptions. It should also be noted that the standard Rayleigh curve that describes staffing profiles embodies the notion of a linear learning function.

At first glance, learning curves may be easily expressed as graphs over time, but this simple depiction will only be accurate when time correlates with cumulative output. As Raccoon discusses, there are biases that mask learning curves. For example, the output of a software process (whether a line of code, a function point, or an object measure) represents a different cost and work in different project phases. Growth biases also affect learning curves, such as the rapid escalation in staff that confuses underlying individual productivities.

There are also process disruptions for periods of time that affect productivity. These can be after a long break (vacation, sickness, holidays, etc.) or other work disruptions. This author distinctly remembers his days in the software development trenches when the first days back after long holiday breaks were spent relearning operating system commands, the keyboard to an extent, and the problem at hand. It definitely took some time to get back to previous levels of productivity. Process bottlenecks may also delay learning, such as when machines pace the work. As Raccoon states: “When biases affect measurement and time does not correlate with cumulative output, then the units of measurement on the x axis must be translated to cumulative output” [Raccoon 1996].

Learning curves are traditionally formulated in terms of the unit costs of production. The most widely used representation for learning curves is called the log-linear learning curve expressed by

$$y = ax^n$$

where a is the cost of the first unit, x is the cumulative output, and n is the learning curve slope. Figure 4.9 shows this relationship. Compare this with Figure 4.8; they are inverse relationships (excepting for the S-shape characteristic) since productivity = 1/unit cost. The slope of a learning curve is related to the learning rate, which describes the scaling of unit cost for every doubling of cumulative output. For example, a learning rate of 75% indicates that the unit cost scales by 75% every doubling of cumulative output. If the unit cost of the 1000th line of code is 60 minutes, then the cost of the 2000th line would be 0.75(60) = 45 minutes. The slope of a learning curve is equivalent to $\log_2$(learning rate). If plotted on a log-log scale, the slope would be directly measurable since the unit cost function would be a line.

Other important learning curve equations per [Raccoon 1996] are:

- Stanford-B: $y = a\,(x + b)^n$
- DeJong: $y = a + bx^n$
- S-curve: $y = a + b\,(x + c)^n$

The log-linear equation is the simplest and most common equation and it applies to a wide variety of processes. It has been shown to model future productivity very effectively. The log-linear curve models the middle portion of the typical learning curve in Figure 4.8 fairly well, but does not account for the beginning and end segments with much accuracy.

In some cases, the DeJong and Stanford-B equations work better. The Stanford-B equation is used to model processes in which experience carries over from one production run to another, so workers start out more productively than the asymptote predicts.

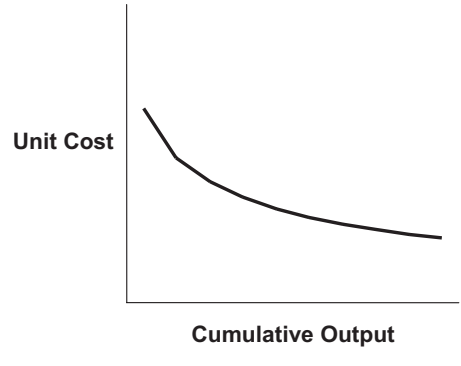

Figure 4.9. Log-linear unit cost function.

The DeJong equation is used to model processes in which a portion of the process cannot improve.

The S-curve equation combines the Stanford-B and DeJong equations to model processes in which both experience carries over from one production run to the next and a portion of the process cannot improve. The S-curve equation often models past productivity more accurately but usually models future productivity less accurately than the other equations. See [Raccoon 1996] for more details.

As it is desirable to maximize learning in an organization, training methods and modes of education come into play. Little or poor training will not provide opportunities to maximize the learning. See the Section 4.7.6, Simulation for Personnel Training, on how simulation itself can be a cost-effective training solution.

4.5.1 Example: Learning Curve Models

Now let us apply some of these learning curve concepts and implement the formulas in system dynamics feedback structures. A summary of these models is in Table 4.3. A demonstration set of learning curve models is provided in *learning curves.itm.* The model in Figure 4.10 uses a log-linear learning curve with a learning factor of 80%. This is the factor for software development estimated by Raccoon, and will be compared later to other data and formulations.

The output for this model shows a 6:1 range in productivity over 500 days (Figure 4.11), which is much greater than intuition and experience dictates. This indicates that some scaling and/or an offset should be employed, but the general shape is reasonable for a learning curve.

The next section shows how this learning function and similar formulations compare to the COCOMO static model experience factors. It also shows a way to mimic the COCOMO multipliers directly using a fixed learning curve as an alternative to the dynamic learning approach described above.

4.5.1.1 Learning Curve Comparison with COCOMO Experience Data

It is best to use empirical data for comparison when assessing the learning curve for software processes. One source of data for this is the COCOMO II project. There are several experience factors in the model. The effort multipliers for these factors are derived from Bayesian statistical analysis of actual reported project data. Some of the

Table 4.3. Learning curve models overview

Purpose: Planning, Training **Scope:** Development Project			
Inputs and Parameters	Levels	Major Relationships	Outputs
• Learning factor • First unit cost • (or) Productivity curve	• Tasks completed	• Unit cost or productivity	• Productivity or productivity multipliers

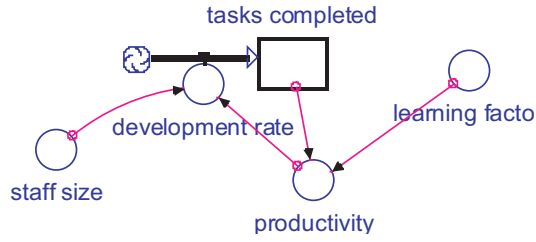

Figure 4.10. Model of log-linear learning curve (learning factor = 0.8).

learning curve formulations will be compared with COCOMO multipliers derived from this data.

Figure 4.12 shows a comparison of the log-linear learning curve with COCOMO experience multipliers using hours of experience as the common unit. There is a mild disparity between the COCOMO points and the continuous learning curve. The COCOMO multipliers exhibit a learning factor of about 0.7 for the first year, 0.8 for the next couple of years, and about 0.82 after three years of experience.

The graph shows reasonable agreement with Raccoon's estimate of an 80% learning rate for software development. The COCOMO multipliers exhibit an S-shaped curve, however, whereby learning is quicker in the beginning (a smaller learning rate) and not as great after several years (e.g., the approximate learning rate of 82%). The S-curve described by $y = a + b(x + c)^n$ can also be fitted to the COCOMO S-shape. This

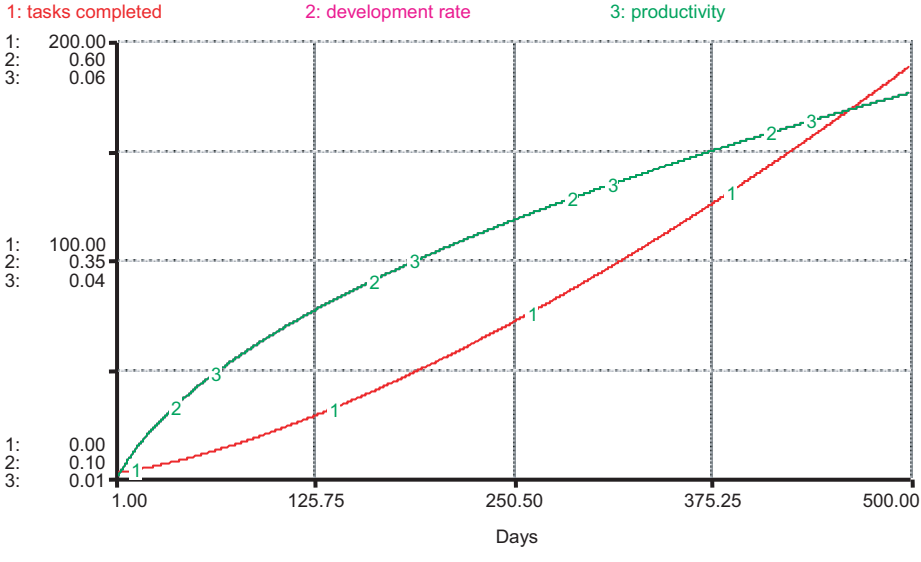

Figure 4.11. Output of log-linear learning curve model.

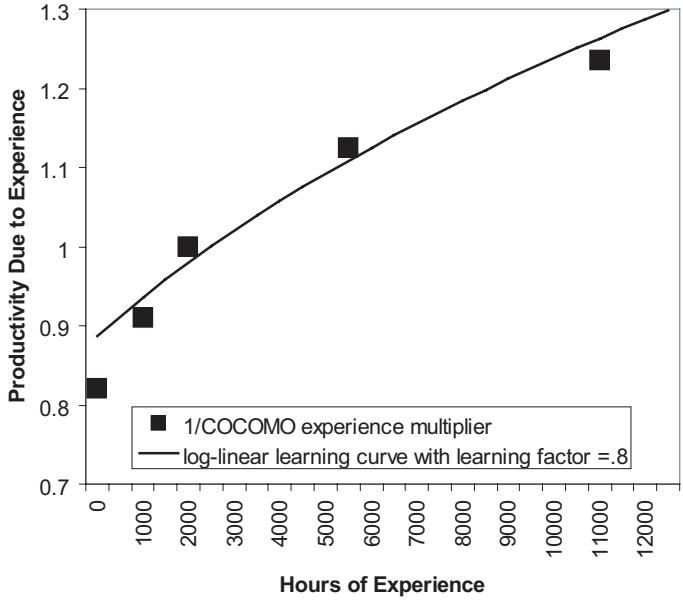

Figure 4.12. Comparison of log-linear learning curve with COCOMO experience multipliers.

mathematical exercise is beyond the scope of this book and is left as an exercise for the interested student.

The modeling implications are that a learning rate of 80% with an initial offset for experience is a fairly good approximation to real data, and that a learning curve is a straightforward formulation for a system dynamics model. The modeler must be aware, though, of the S-shaped characteristics that would produce relatively larger errors in the short term (about one year) and very long term (beyond about 6 years). If only those time portions are of interest, then use the appropriate learning rates instead. For example, use a 70% learning rate if the time horizon is a year or less.

After applying an offset for accumulated experience to the log-linear learning curve model in Figure 4.10, its new corresponding output is in Figure 4.13. The shape is valid for a learning curve and covers the expected region of productivity.

These results against the COCOMO experience factors with learning curves have been corroborated by others. In [Eickelman et al. 2002] the authors used simulation and benchmarking for quantitative control of process changes, and explored productivity issues. Different productivity learning curves were evaluated and compared to COCOMO II factors. They also found that a curve with an 80% learning rate is the best match to COCOMO data. The regions in which the curves diverged were the same as the results above. They found that using a learning curve helped in benchmarking from year to year and in evaluating longer-term trends and cumulative effects of training and new technologies. A simulation model was useful for evaluating short-term impacts to productivity in a 2 months to 6 year time horizon.

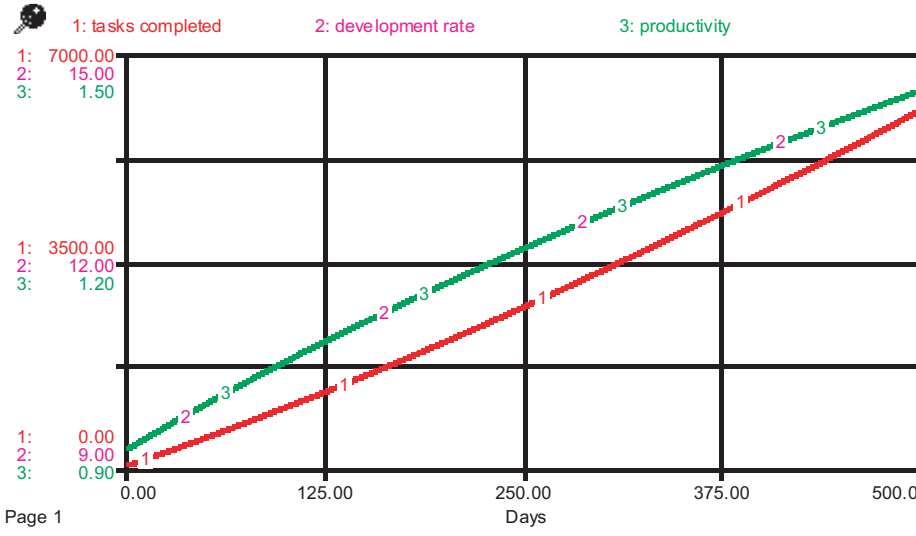

Figure 4.13. Learning curve model output.

4.5.1.2 *Fixed Learning Functions*

Alternatively, a table function could be generated to mimic the COCOMO multipliers. Figure 4.14 shows a function that embodies a productivity multiplier equal to the inverse of the COCOMO effort multipliers. The modeler must be aware, though, of the uncovered region for large time values, as opposed to the log-linear formulation that holds for all time.

A fixed learning curve was used in early system dynamics modeling of software processes at NASA Jet Propulsion Laboratories (JPL) [Lin et al. 1997]. It was one of the first projects to enhance the Abdel-Hamid model, and Chi Lin led the modeling. They employed a fixed learning curve that was calibrated to their typical project length of approximately two years. Figure 4.15 shows the resulting curve as a simple multiplier of productivity. It assumes full-time dedication to the project and approximates the aggregation of COCOMO experience factors. Such a curve will not be the same for a 3-month development project as for a 2-year project. It should be used with caution and recalibrated for different environments.

Learning curves are represented in several other applications in this book. A good example is the use of different learning curves for new language training (see the Chapter 5 modeling example on reuse and high-level languages).

4.6 TEAM COMPOSITION

A hybrid process involves elements of both agile and plan-driven approaches. One of the people issues for hybrid processes is how to compose the teams in terms of skill

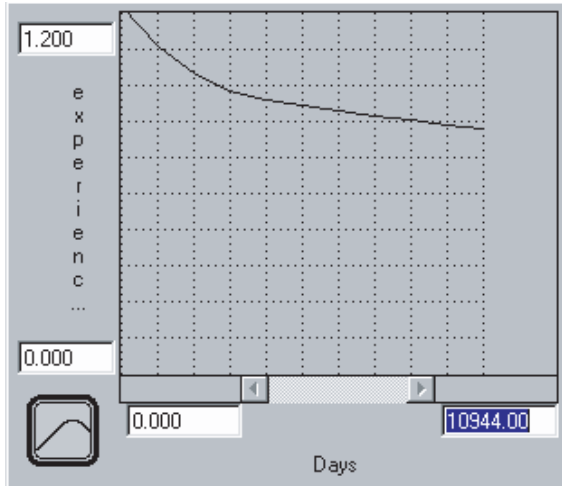

Figure 4.14. Table function for productivity multiplier based on COCOMO experience factor (experience multiplier vs. days).

sets and responsibilities. The following section is based on [Madachy et al. 2007], where simulation is used to find the optimum number of agile people to match anticipated change traffic.*

4.6.1 Example: Assessing Agile Team Size for a Hybrid Process

New processes are being assessed to address modern challenges for Software-Intensive Systems of Systems (SISOS), such as coping with rapid change while simultaneously assuring high dependability. A hybrid agile and plan-driven process based on the spiral life cycle has been outlined to address these conflicting challenges by rapidly fielding incremental capabilities in a value-based framework. A system dynamics model has been developed to assess the incremental hybrid process and support project decision making. It estimates cost and schedule for multiple increments of a hybrid process that uses three specialized teams, and also considers the mission value of software capabilities. It considers changes due to external volatility and feedback from user-driven change requests, and dynamically reestimates and allocates resources in response to the volatility. Deferral policies and team sizes can be experimented with, and it includes trade-off functions between cost and the timing of changes within and across increments, length of deferral delays, and others. We illustrate how the model can be used to determine optimal agile team size to handle changes. Both the hybrid

*This is classified as a people application instead of a project application because the primary state variables of interest are the number of people, especially the agile team. Project staffing applications in Chapter 7 use tasks, effort, or other state variables, and staffing needs are derived from those (though there is strong overlap between staffing and people applications).

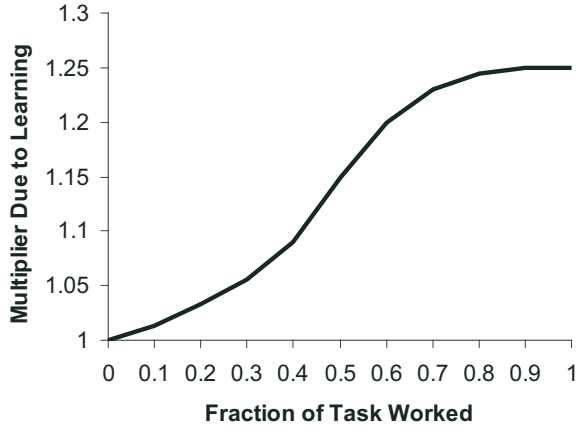

Figure 4.15. JPL learning function (two-year project).

process and simulation model are being evolved on a very large-scale incremental SISOS project and other potential pilots. See Table 4.4 for a summary of the model.

4.6.1.1 Introduction and Background

Our experiences in helping to define, acquire, develop, and assess 21st century SISOS have taught us that traditional acquisition and development processes do not work well on such systems [Boehm et al. 2004, Boehm 2005]. At the University of Southern California (USC) Center for Systems and Software Engineering (CSSE) we are using simulation modeling to help formulate and assess new processes to meet the challenges of these systems.

The systems face ever-increasing demands to provide safe, secure, and reliable systems; to provide competitive discriminators in the marketplace; to support the coordination of multicultural global enterprises; to enable rapid adaptation to change; and to help people cope with complex masses of data and information. These demands will cause major differences in current processes [Boehm 2005].

The USC team and others have been developing, applying, and evolving new processes to address SISOS challenges. These include extensions to the risk-driven spiral model to cover broad (many systems), deep (many supplier levels), and long (many increments) acquisitions needing rapid fielding, high assurance, adaptability to high change traffic, and complex interactions with evolving commercial off-the-shelf (COTS) products, legacy systems, and external systems.

The distinguishing features of an SOS are not only that it integrates multiple independently developed systems, but also that it is very large, dynamically evolving, and unprecedented, with emergent requirements and behaviors and complex sociotechnical issues to address. Thus, we have developed a system dynamics model because the methodology is well suited to modeling these dynamic phenomena and their interactions.

Table 4.4. Hybrid incremental process model overview

Purpose: Planning, Process Improvement
Scope: Multiple, Concurrent Increments

Inputs and Parameters	Levels	Major Relationships	Outputs
• Increments • Increment overlap • Baseline capabilities per increment • Agile team size • Change policies • Volatility profile • Capability flags • Volatility multiplier • Life-cycle timing multiplier • Change analysis effort • Construction effort distribution • Increment values • Field issue metrics • Schedule per increment • Overall costs per increment	• Team sizes Agile team Developer team V&V team • Software capabilities per increment Changes Required Developed V&V'ed • Effort per increment	• Volatility tradeoffs • Change analysis dynamics • Effort and schedule algorithms • Staffing change algorithms	• Staffing profiles per team per increment

4.6.1.1.1 Previous Simulation Work. No previous work in software process modeling has focused on hybrid processes for SISOS development. A few efforts have addressed incremental (or iterative) development and the effects of requirements changes. Software requirements volatility and change control dynamics was investigated in [Ferreira 2002] and [Ferreira et al. 2003]. Data in that research showed an average of 32% requirements volatility in over 200 projects, which was captured in their model. Iteration management as a way to reduce cycle time in the midst of change was addressed in [Ford, Sterman 2003], who modeled the 90% syndrome and impact of concealing requirements changes. The use of fire-fighting techniques to handle late changes is described in [Repenning 2001], which illustrates how late rework leads to perpetual firefighting in a multiproject development environment. Both [Ford, Sterman 2003] and [Repenning 2001] addressed high technology applications, but not in the specific context of software projects.

A few efforts have simulated incremental or iterative software development. The research in [Tvedt 1996] modeled concurrent incremental development and the impact of inspections. The cost and schedule impacts of different increment options were assessed with the model. Another incremental development model to assess cost and schedule was developed in [Sycamore 1995], though both of these models were limited in their cycle variability since they used replicated structures to represent the different increments. Iterative development was modeled with system dynamics in [Powell

et al. 1999] and [Powell 2001] to assess both concurrent software engineering and staged delivery as methods to improve cycle time. Requirements changes were included in the iterative model in [Powell 2001] and were based on substantial empirical data. However, none of these efforts considered personnel makeup to best handle changes.

A qualitative model of agile processes is described in [Fernández-Ramil et al. 2005] but this is the first known instance of modeling with system dynamics applied to either hybrid or agile processes.

The model presented here builds on the concepts of requirements volatility and incremental or iterative development in previous research, but it goes a step further by showing the effects of a hybrid process used in incremental development to move away from firefighting to a more controlled process for accommodating changes quickly while balancing software cost, schedule, and value issues.

4.6.1.1.2 The Scalable Spiral Model. The outlines of a hybrid plan-driven/agile process for developing an SISOS product architecture are emerging. It is a risk-driven balance of agility and discipline [Boehm, Turner 2004]. In order to keep SISOS developments from becoming destabilized due to large amounts of change traffic, it is important to organize development into plan-driven increments in which the suppliers develop to interface specs that are kept stable by deferring changes, so that the systems can plug and play at the end of the increment. But for the next increment to hit the ground running, an extremely agile team needs to be concurrently and continuously monitoring the market, competition, and technology, doing change impact analysis, refreshing COTS, and renegotiating the next increment's prioritized content and the interfaces between the suppliers' next-increment interface specs.

The spiral model was introduced in 1986 and later elaborated for WinWin extensions [Boehm et al., 1998]. It has continued to evolve to meet the needs of evolving development processes. We have been converging on a scalable spiral process model for SISOS that, for partial implementations to date, has scaled well from small e-services applications to super-large defense systems of systems and multienterprise supply chain management systems.

Figure 4.16 shows a single increment of the development and evolution portion of the model. It assumes that the organization has developed:

- A best-effort definition of the system's steady-state capability
- An incremental sequence of prioritized capabilities culminating in the steady-state capability
- A feasibility rationale providing sufficient evidence that the system architecture will support the incremental capabilities, that each increment can be developed within its available budget and schedule, and that the series of increments create a satisfactory return on investment for the organization and mutually satisfactory outcomes for the success-critical stakeholders

As seen in Figure 4.16, the model is organized to simultaneously address the conflicting challenges of rapid change and high assurance of dependability. It also addresses

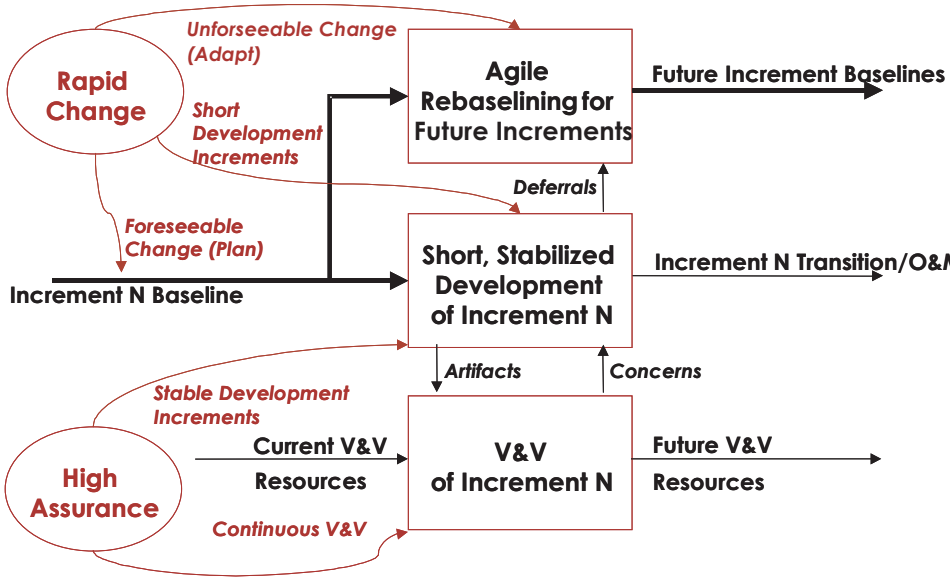

Figure 4.16. The scalable spiral process model: increment activities.

the need for rapid fielding of incremental capabilities with a minimum of rework, and the other trends involving integration of systems and software engineering, COTS components, legacy systems, globalization, and user value considerations.

The hybrid process uses a three-team cycle (lean, plan-driven, stabilized developers; thorough V&Vers; and agile, proactive rebaseliners) that plays out from one increment to the next.

The need to deliver high-assurance incremental capabilities on short fixed schedules means that each increment needs to be kept as stable as possible. This is particularly the case for very large systems of systems with deep supplier hierarchies in which a high level of rebaselining traffic can easily lead to chaos. The risks of destabilizing the development process make this portion of the project into a waterfall-like, build-to-specification subset of the spiral model activities. The need for high assurance of each increment also makes it cost-effective to invest in a team of appropriately skilled personnel to continuously verify and validate the increment as it is being developed.

However, "deferring the change traffic" does not imply deferring its change impact analysis, change negotiation, and rebaselining until the beginning of the next increment. With a single development team and rapid rates of change, this would require a team optimized to develop to stable plans and specifications to spend much of the next increment's scarce calendar time performing tasks much better suited to agile teams.

The appropriate metaphor for addressing rapid change is not a build-to-specification metaphor or a purchasing-agent metaphor but an adaptive "command–control–intelligence–surveillance–reconnaissance" (C2ISR) metaphor. It involves an agile team performing the first three activities of the C2ISR "Observe, Orient, Decide, Act"

(OODA) loop for the next increments, while the plan-driven development team is performing the "Act" activity for the current increment. These agile activities are summarized below:

- Observing involves monitoring changes in relevant technology and COTS products, in the competitive marketplace, in external interoperating systems, and in the environment; and monitoring progress on the current increment to identify slowdowns and likely scope deferrals.
- Orienting involves performing change impact analyses, risk analyses, and tradeoff analyses to assess candidate rebaselining options for the upcoming increments.
- Deciding involves stakeholder renegotiation of the content of upcoming increments, architecture rebaselining, and the degree of COTS upgrading to be done to prepare for the next increment. It also involves updating the future increments' feasibility rationales to ensure that their renegotiated scopes and solutions can be achieved within their budgets and schedules.

A successful rebaseline means that the plan-driven development team can hit the ground running at the beginning of the "Act" phase of developing the next increment, and the agile team can hit the ground running on rebaselining definitions of the increments beyond.

As much as possible, usage feedback from the previous increment is not allowed to destabilize the current increment, but is fed into the definition of the following increment. Of course, some level of mission-critical updates will need to be fed into the current increment, but only when the risk of not doing so is greater that the risk of destabilizing the current increment.

4.6.1.2 Model Overview

The primary portion of the system dynamics model diagram showing increment activities and the teams is in Figure 4.17. It is built around a cyclic flow chain for capabilities and uses arrays to model multiple increments. The flow chains for the increment activities show multiple layers of levels and rates; these identify array elements that correspond to the increments. Thus, the flow chain and its equations are arrays of five to model five increments (this preset number can be changed to model more or fewer increments). Note that multiple flows that appear to be flowing into the same level may be flowing into different levels with different array indices, depending on the logic of the increment allocation of capabilities.

Unanticipated changes arrive as aperiodic pulses via the *volatility trends* parameter. This is how they actually come on the projects as opposed to a constant level of volatility over time. The user can specify the pulses graphically (see the input for volatility profile in Figure 4.18) or use formulas. The *capability volatility rate* will flow the changes into the corresponding increment for the current time. From there they arrive in the level for *capability changes* and are then processed by the agile rebaselining team. They analyze the changes per the *average change analysis effort* parameter.

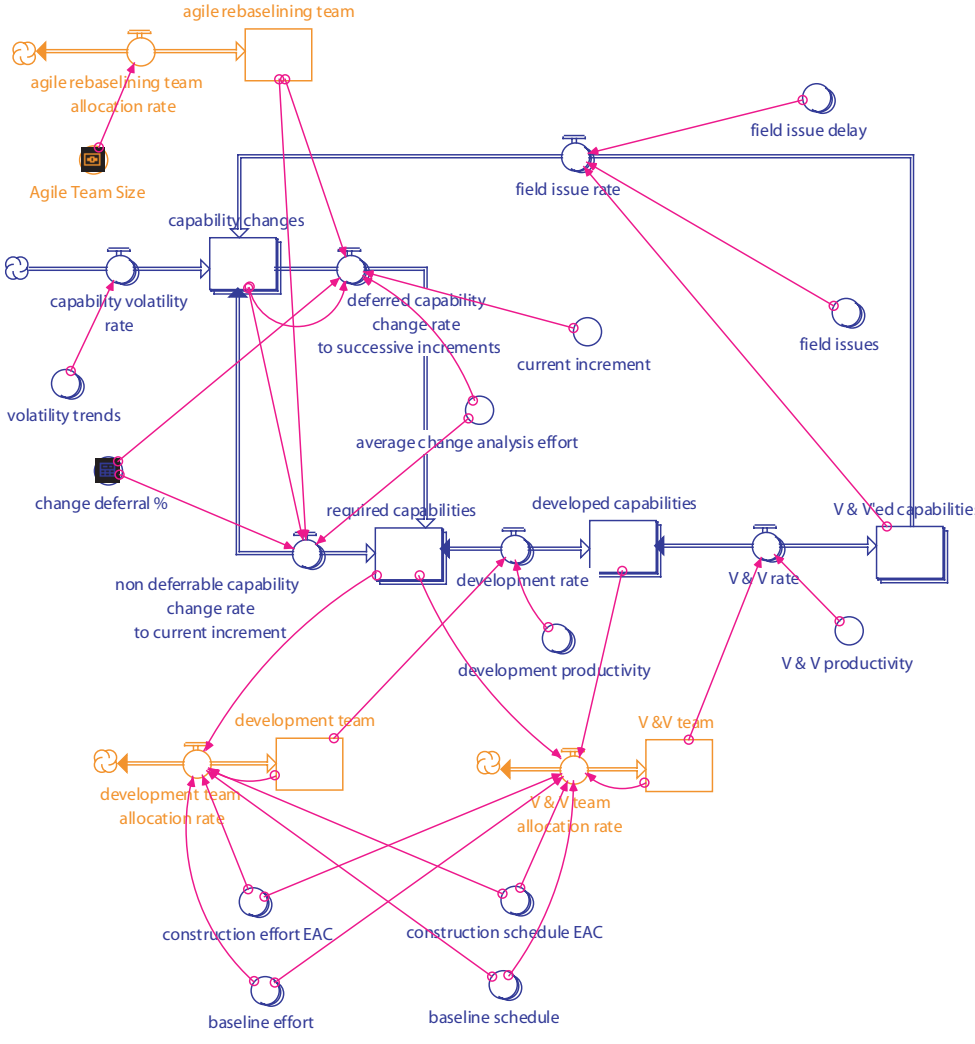

Figure 4.17. Model diagram.

Their overall productivity is a function of the *agile team size* (as specified by the user in Figure 4.18) and the average analysis effort.

The *change deferral %* is a policy parameter used to specify the percentage of changes that must be deferred to later increments via the *deferred capability change rate to succeeding increments* to the *required capabilities* for the appropriate increments. The remaining ones are nondeferrables that flow into the *required capabilities* for the current increment via the rate *non deferrable capability rate change to current increment*. The deferral policy parameter is also shown in the inputs in Figure 4.18.

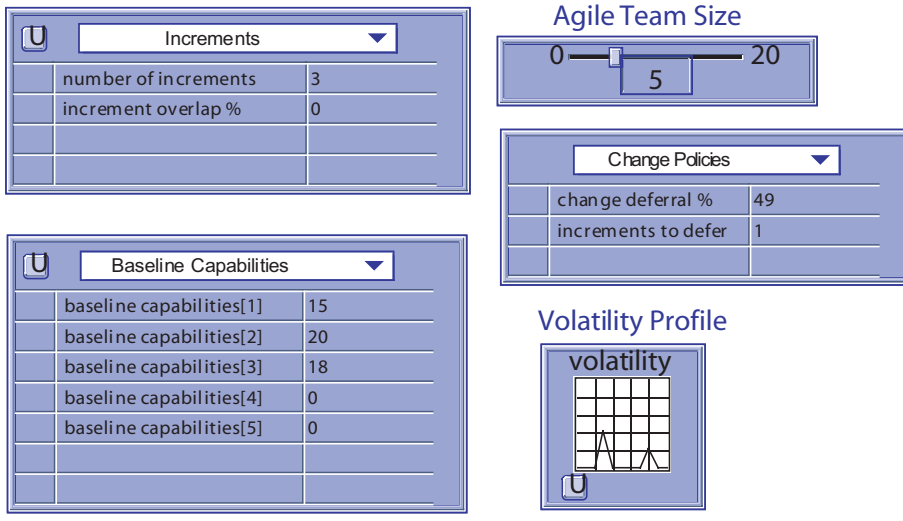

Figure 4.18. Simulation inputs.

The two arrayed flows between *capability changes* and *required capabilities* are complicated multiplexers that cannot be fully visualized on the diagram. On both paths, there are five flows (or pipes) between them that capabilities may go through and the capabilities can move between the different increment flows. For different reasons, capabilities may be assigned, deferred, or delayed to any of the five increments, and the full set of possible paths cannot be seen.

When an increment starts, the *required capabilities* are developed by the development team at the *development rate* and flow into *developed capabilities* (all using the flow chain array index corresponding to the proper increment).

Similarly, the *developed capabilities* are then picked up by the V&V team for their independent verification and validation. They do their assessment at the *V&V productivity rate* and the capabilities flow into *V&V'ed capabilities.*

The rates in the flow chain between *capability changes, required capabilities, developed capabilities,* and *V&V'ed capabilities* are all bidirectional. There is a provision for capabilities to be "kicked back" or rejected by the various teams and sent back up the chain. For example, there are times when the developers have major concerns about a new capability and send it back to the rebaselining team. Likewise, the V&V team might find some serious defects that must be reworked by the developers.

Finally there are user-driven changes based on field experience with the system. These are identified as *field issues* that flow back into the *capability changes* per the *field issue rate* at a constant *field issue delay* time. The *field issues* parameter represents the amount of concern with the fielded system and accounts for a primary feedback loop.

The agile baselining team is shown in the top left of the diagram. The size of the team can be specified as a constant size or a varying number of people over time via

the inputs in Figure 4.18. The *agile rebaselining team allocation rate* flows people in or out of the team to match the specified team size over time.

The development and V&V teams are shown at the bottom. Their allocation rates are based on the construction effort and schedule for the required capabilities known to date. Currently, the productivities and team sizes for development and V&V are calculated with a Dynamic COCOMO [Boehm et al. 2000] variant. They are equivalent to COCOMO for a static project (the converse situation of this model context) and are continuously recalculated for changes. However, this aspect of the model whereby the team sizes are parametrically determined from size and effort multipliers will be refined so that constraints can be put on the development and V&V staff sizes. See Section 4.6.1.2.2 for more details on the staffing algorithms.

4.6.1.2.1 Trade-off Functions. There are several functional relationships in the model that effect trade-offs between deferral times and cost/schedule. For one, it is costlier to develop software when there is a lot of volatility during the development. If required capabilities are added to an increment being developed, the overall effort increases due to the extra scope as well as the added volatility. The effort multiplier in Figure 4.19 is used to calculate the construction effort and schedule based on a volatility ratio of total required capabilities to the baseline capabilities. This example shows a simple linear relationship, though the graphical construct allows it to be user defined and become nonlinear (as in Figure 4.20 described next).

The volatility factor in Figure 4.19 is an aggregate multiplier for volatility from different sources. It works similarly to the platform volatility multiplier in COCOMO II, the USC CSSE software cost estimation model. The difference in this context is that there may be many more sources of volatility (e.g., COTS, mission, etc.). This multiplier effect only holds for an increment when changes arrive midstream. If new

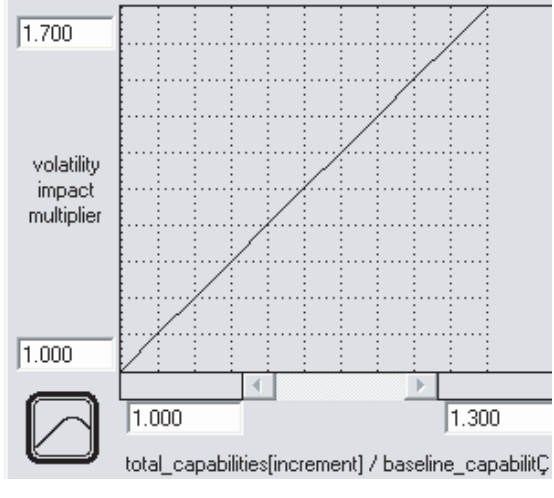

Figure 4.19. Volatility effort multiplier.

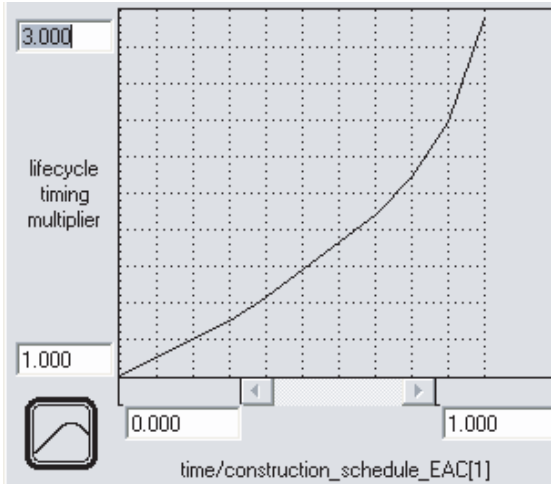

Figure 4.20. Life-cycle timing effort multiplier.

changes are already in the required capabilities when an increment starts, then it has no effect.

Additionally, the later a new capability comes in during construction, the higher the cost to develop it. This is very similar to the cost-to-fix defects, for which the costs increases exponentially. Figure 4.20 shows the life-cycle timing multiplier based on a ratio of the current time to the entire increment schedule.

Under normal circumstances, there is an additional cost of delaying capabilities to future increments because there is more of a software base to be dealt with and integrated into. Therefore, we increase the cost of deferring to future increments by an additional 25% relative to the previous increment (this parameter is easily changed).

4.6.1.2.2 Dynamic Resource Allocation. In response to changes in the capabilities, the model calculates the personnel levels needed for the new increment size and interpolates for the amount of work done. Part of the structure for continuous calculation of staffing levels is shown in Figure 4.21, where EAC stands for "estimate at completion." Baseline effort and schedule refer to the original plan without any volatility. The parameters for construction effort and construction schedule are the ones used to determine the staffing allocation rates shown at the bottom of Figure 4.17.

An example of the delta staffing for the development team when new capability changes come into the system is:

development team allocation rate = (development_effort_fraction
· construction_effort_EAC(increment)/development_schedule_fraction
· construction_schedule_EAC(increment)) – development_effort_fraction
· baseline_effort(increment)/(development_schedule_fraction
· baseline_schedule(increment)

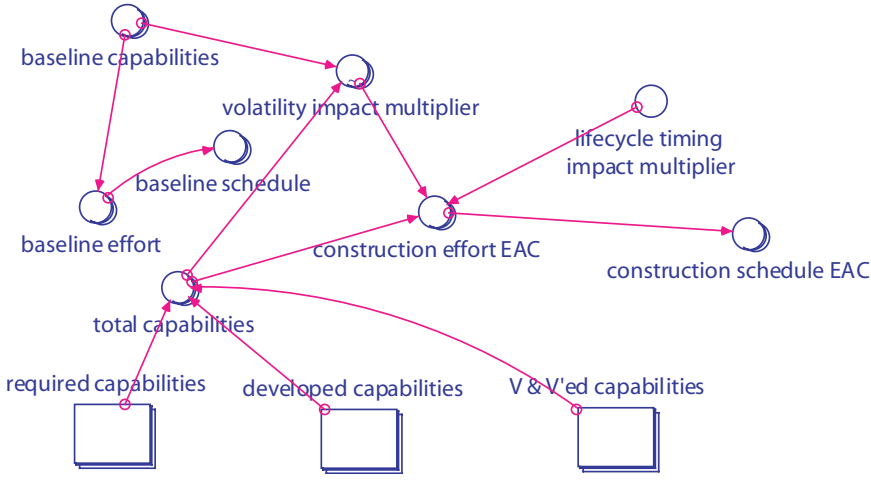

Figure 4.21. Staffing calculation parameters.

The development effort and schedule fractions are the calibrated portions of effort and schedule for development with respect to all of construction, and the V&V allocation algorithm works similarly. If the increment has just started, then the interpolated staffing level will be closer to the higher level needed for the new EAC. If the increment is mostly done, then it does not make sense to increase staff to the EAC level because almost all the work is done anyway.

On the SISOS project we are primarily applying the model to, there are already over 2000 software development people on board, and most contractors have already hired their steady-state levels of personnel. Thus, it is reasonable to assume a nearly constant level of construction staffing with no major ramp-ups, and a step-function staffing profile that approximates labor with average levels and nonoverlapping phases.

The step-function staffing profile is improved with a Rayleigh curve staffing version of the model that intrinsically changes the staffing when changes occur, with no interpolation necessary. It also assumes overlap of activities, and the starting point of V&V can be specified with respect to development. For illustration here, we will continue to use the step-function version since the results are generally easier to follow and interpret.

4.6.1.2.3 PARAMETERIZATIONS. Since this is a macro model for very large systems, a capability is a "sky level" requirement measure. It is defined as a very high-level requirement that we have made equivalent to 10 KSLOC for the purpose of estimation. The construction effort and schedule is currently calculated with a Dynamic COCOMO approach using the COCOMO II.2000 calibration [Boehm et al. 2000].

The volatility impact multiplier is an extension of COCOMO for the SISOS situation. It is extrapolated from the current model and partially based on expert judgment. Other parameterizations relying on expert judgment include the average change analy-

sis effort, life-cycle timing multiplier, and amount of field issues. They will be updated based on empirical data we are collecting.

4.6.1.3 Sample Scenario and Test Case Results

We have put the model through multiple scenarios to assess project options. Here, we demonstrate a seemingly simple scenario and show how it plays out to be a rather complex trade-off situation. The scenario being demonstrated is for developing two successive increments of 15 capabilities each, with a nondeferrable change coming midstream that has to be handled. The agile team size is varied from two to twenty people in the test cases.

A capability change comes in at month eight and is processed by the agile team. The change is nondeferrable as it needs to be in Increment 1. An illustration of how the system responds to such a volatility pulse in Increment 1 is shown in Figure 4.22. In the figure legend, "[1]' refers to the increment number 1. An unanticipated set of changes occurs at month 8, shown as a *volatility trend* pulse. The changes immediately flow into the level for *capability changes,* which then starts declining to zero as an agile team works it off per the average change analysis effort of four person-months. The change is nondeferrable and it becomes incorporated into Increment 1, so the *total capabilities* for the increment increases. As the new capabilities become required for Increment 1, the development staffing responds to the increased scope by dynamically adjusting the team size to a new level.

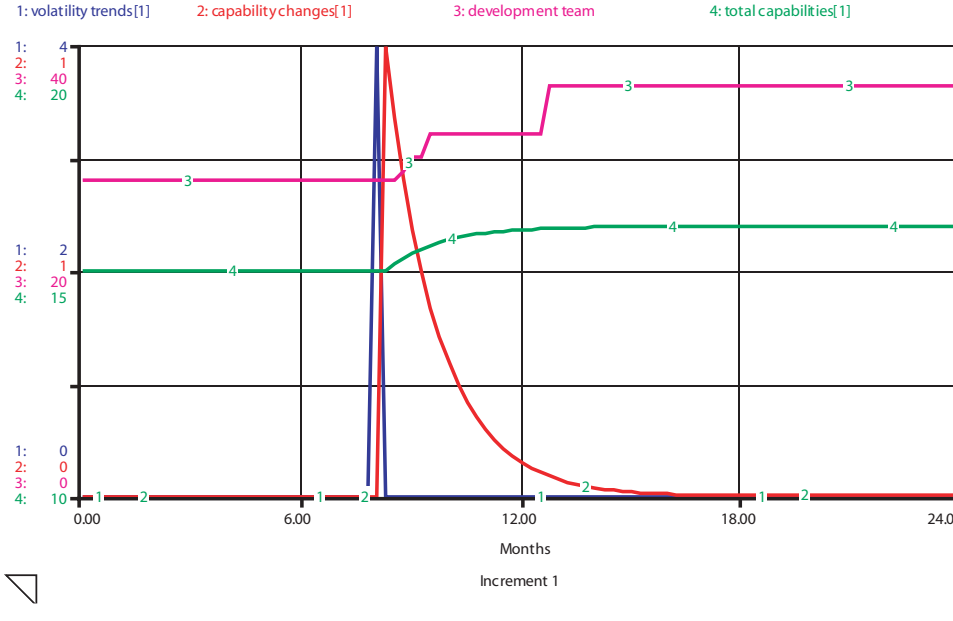

Figure 4.22. System response to volatility—Increment 1.

However, the different team sizes will analyze the change at different rates. Figures 4.23–4.25 show the test cases results for varying the agile team size. If the team size is too small, then it will not even make it into Increment 1. The agile team cannot even process the change in time for Increment 1 with two to four people. In these cases, the new capability for Increment 1 is processed too late and goes into Increment 2 instead. But with six people on the agile team it can make it in time. The larger team size will process the change and incorporate it faster; hence, the effort and schedule for Increment 1 improves with an agile team size of six or higher.

A summary of the resulting dynamics for these cases are:

- Two agile people take the longest time to process the change. The capability is ready for implementation late in Increment 2 and incurs a substantial effort penalty per the lifecycle timing effort multiplier. Its cost is greater than if the capability were ready at the start of Increment 2. Increment 2 also incurs the volatility multiplier effect and costs more than its original baseline. Increment 1 executes to its baseline plan.
- Four agile people process the change faster, but still not in time for Increment 1. The Increment 2 life-cycle timing impact is not as great as the case for two people. Increment 1 again matches its original baseline plan.
- Six people can process the change fast enough to get it into Increment 1, but it comes in late and incurs the late life-cycle timing effort multiplier. Increment 2 does not vary from its baseline
- Eight people are faster than six, so the Increment 1 losses are reduced and Increment 2 is still at its baseline.
- Ten people similarly improve upon the team of eight. Increment 1 construction effort is reduced relative to the team of eight because the change comes in sooner. Increment 2 remains unchanged.
- As the agile team grows from ten to twenty people, the impact of late life-cycle changes decreases less than the added effort due to more agile people. With larger agile team sizes, the effect of the life-cycle timing effort multiplier in Figure 4.20 moves from the steep portion at the top down to the shallow part where changes come early in the increment.

Figure 4.23 shows the components of effort as a function of the agile team size. The effort for the constant-size agile team increases linearly as its size grows. It is the smallest relative component of effort when under ten people. The Increment 1 construction effort (development + V&V) at two and four agile people is the baseline effort since the capability change did not reach it. It is highest cost at six people due to the late life-cycle effort multiplier effect, and that penalty is reduced at eight and ten people since it comes in sooner. Above ten people, the total cost rises slowly due to the agile team size increasing and impact of lateness decreasing, as previously described.

Roughly converse effects are shown in the effort function for Increment 2. At two and four people, the cost of Increment 2 is very high because the small agile team takes a long time to analyze the change. It comes in latest with the smallest team incurring a

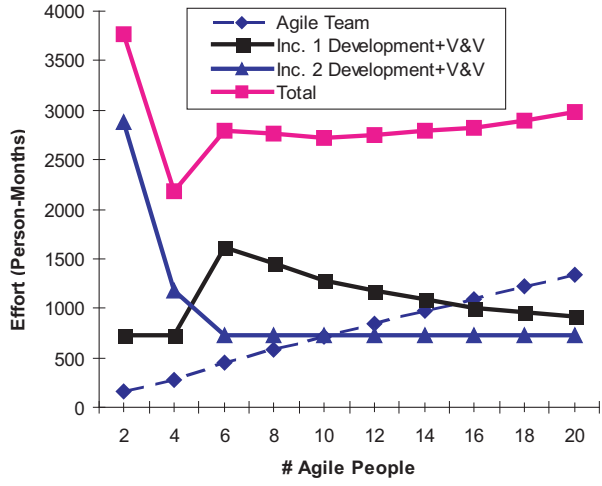

Figure 4.23. Effort versus agile team size.

very high penalty for being late in the life cycle. Six through twenty people can process the change before Increment 2 starts, so the effort is the same for those cases. The Increment 2 effort exceeds its original baseline due to the added volatility of the single capability.

Figure 4.24 shows the test case schedule results for both increments and overall. They are analogous to the effort functions shown in Figure 4.23 for the two increments. Their shape mirrors the effort curves at approximately the 1/3 power. This em-

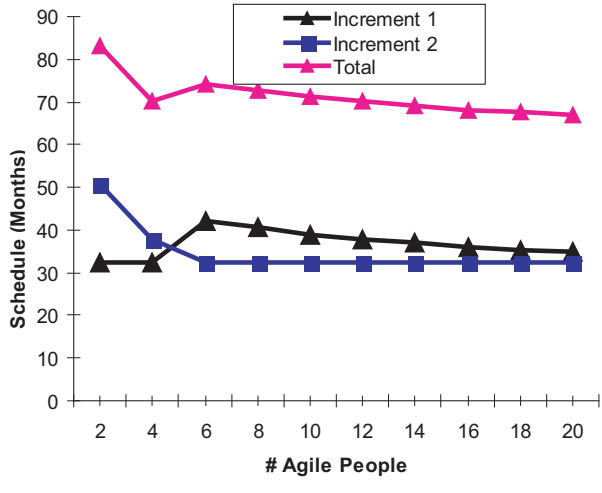

Figure 4.24. Schedule versus agile team size.

pirical correlation between software effort and schedule is described in [Boehm et al. 2000].

Investigating Figure 4.23 and Figure 4.24 may lead to the erroneous conclusion that four people is the optimum since that produces the least overall cost and schedule. The total effort for four agile people may look promising since the change was effectively deferred and did not incur life-cycle timing losses. However, the bigger picture must consider the mission value losses incurred by the smaller teams.

4.6.1.3.1 SOFTWARE VALUE CONSIDERATIONS. These results are incomplete without casting them in a value-based framework. The model outputs described so far cover development cost and schedule impacts, but not the mission value of the software intended for the increments. Our project context of an extremely large government defense project places value on the utility of mission capabilities in terms of war-time performance measures. However, the value-based technique also is applicable to commercial projects where the software utility is more directly quantified in terms of monetary return on investment (ROI). See the business case model in Chapter 5 for process and product decisions using business value measures such as market share, sales, and ROI.

The mission capability values will be simplified using the conservative assumption that they are at least equivalent to their cost of development. Presumably in any software venture, the desired capabilities are worth at least their cost, otherwise there is no reason to take on the project, though there are exceptions to knowingly lose money on development as an investment to gain better chances for larger future gains (e.g., internal product research and development for a potentially large external market).

In our project case, the required capabilities are direly needed to address real battle threats, and they are carefully planned out in time. Real mission value will be lost if they are not delivered on time. Our assumption of their value being equal to development cost is unquestionably very conservative; in reality, their utility values are judged to be several times more. Their utility does not expire 100% at the planned time of delivery, but for simplification we will assume so in our analysis and still demonstrate a valid picture of the situation. Our two simplifying assumptions also balance out each other a bit, by underestimating the mission value and possibly overestimating the full discrete loss at intended delivery time. Now we have accounted for a discrete mission value loss in Increment 1, but there are even more time-dependent losses to account for since Increment 2 stretches out. This analysis can be further refined to account for the loss of value as a diminishing function over time, and the results will vary a bit.

Figure 4.25 shows the value-based analysis for handling the capability change in terms of all costs, and can be considered a minimum view of the actual value loss per our conservative assumption previously described. With a more global value consideration, it is evident that four people are no longer the optimum. According to these results, a larger team size is optimum and the losses decrease very slightly when increasing the team size up to ten people. The region of slightly rising effort between ten and twenty people produces the sweet spot cost minimum at ten people.

These results account for the life-cycle timing multiplier, volatility multiplier, increment delay losses, and a quantification of software capability value. The model

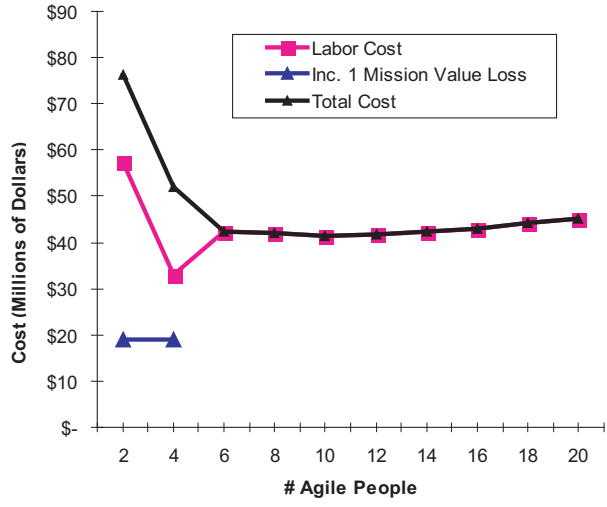

Figure 4.25. Cost versus agile team size.

shows that a sufficient level of agile rebaseliners is necessary, or the cost and schedule for the project increases substantially. The losses are even worse when considering the mission value. Enough people must be on board and productive enough to analyze the changes in a timely manner. Otherwise, there could be a backlog of work to worry about at the beginning of a later increment that could have been resolved earlier by the agile team, or there are other losses. If we further refine upward our conservative estimate of the mission value, the results will lead to the same decision.

Without explanation of these dynamic effects across increments, this apparently simple case of a single change may seem confounding at first with the multiple maxima and minima in the results. When adding the consideration of mission value, the results become even more cogent, explainable, and grounded in practicality. Another corresponding example is to show the results for a single change destined for Increment 2 while holding all the experimental parameters the same. The results are similarly complex, and we are going forward with scenario permutations consisting of multiple capability changes.

4.6.1.4 Conclusions and Future Work

Processes need to be rethought for current and upcoming SISOS, and the outlined hybrid process based on the scalable spiral model appears to be an attractive option. The dynamic model will help to further refine the hybrid process and determine optimized variants for different situations.

Traditionally, it has been thought that agile processes are not suitable for very large software development projects, particularly ones with high dependability requirements. However, this approach addresses concerns by retaining large pockets of stability on the construction teams for development and V&V. We believe this hybrid ap-

proach would be applicable to programs across the size spectrum from small to very large. This is another aspect that we can eventually address with the model by descaling the simulated project and even changing its domain context.

This second major iteration of the model provides interesting results. It shows that if the agile team does not do their work rapidly enough, then developers will have to do it at a higher cost due to changes late in the life cycle, and mission losses may also occur. Further experiments are underway to vary the deferral percentages and perturbation smoothing policies, include rework, and constrain the staff sizes for development and V&V.

Both the hybrid process and the model will be further analyzed and evolved. Various improvements in the model have already been identified and are briefly discussed below, but further changes will come from users of the model and additional empirical data will be used to calibrate and parameterize the model.

The set of test cases we demonstrated varied agile team size, but other dimensions to vary are the deferral percentage and perturbation smoothing policies. Additionally, we are simulating all five increments with volatility occurring in more than one increment in experiments.

This version of the model uses step function staffing profiles that adjust dynamically to changes. Another version uses Rayleigh curves for more realistic staffing patterns that adjust on the fly to midstream changes. These models will be integrated to allow the user to specify the type of staffing.

In the current test cases, only the optimum personnel levels are used for development and V&V but, in reality, there may be staffing constraints. The model will be refined so users can constrain the development and V&V staff sizes. Another set of tests will compare trade-offs between different agile team staffing policies (e.g., level of effort vs. demand driven).

Patterns of changes and change policies will be experimented with. We will vary the volatility profiles across increments and demonstrate kickback cases for capabilities flowing back up the chain from the developers or V&V'ers. Additionally we will model more flexible deferral policies across increments to replace the current binary simplification of allocating changes to the current or next increment.

We are implementing an increment "balancing policy" so that large perturbations for an increment can be optionally spread across increments. The user can specify the number of increments and the relative weighting of capability across them. This balancing policy will tend to smooth large fluctuations in staffing and minimize costly ramp-ups (but always traded off against lost mission value).

Value-based considerations should be part of the overall process analysis. We showed that accounting for the mission/business value of software capabilities made the results more practical, coherent, and explainable in terms of finding trade-off sweet spots. We will refine the value-based considerations to quantify the time-specific value of capabilities instead of assuming full discrete losses.

Parts of the model have been parameterized based on actual empirical data, but not the change traffic. We will be getting actual data on volatility, change traffic trends, and field issue rates from our USC affiliates and other users of the model. Data for the change analysis effort and volatility cost functions will also be analyzed.

After we get change data to populate the model and make other indicated improvements, we plan to use it to assess increment risk for a very large-scale SISOS program. It will also be used by contractors on the program in addition to our own independent usage to assess process options.

We also plan to apply it to other projects we are involved with, and the model will be supplied to our USC-CSSE industrial affiliates for assessing and improving their processes. Some of these users will also provide us with additional empirical data for parameterizing the model.

4.7 OTHER APPLICATION AREAS

This section is a special case for the applications chapters. There are many important areas of people issues not covered by examples earlier in this chapter. Some of these are addressed below to help "prime the pump" and overview areas for which no major models were available or only preliminary efforts have been made. Therefore, these sections are not supported with specific models except for some small demonstrations and/or causal loops. All of them provide fodder for future people-modeling applications.

4.7.1 Motivation

Motivation can have tremendously varying impact over time. It may even eclipse the effects of skill and experience on projects. Motivation may wax and wane due to a multitude of factors and impact productivity either way. A developer may or may not be interested in the application, he/she may have a very good or poor relationship with superiors, and management might be responsible for creating conditions for high or low motivation. For example, if a project is perceived to be not particularly important to the company, then developers will likewise not place too much emphasis on their performance. If there are impending organizational changes or just rumors to that effect, then some people will go into a "wait-and-see" mode and not produce much until their place in the organization is known. On the other hand, if there is a competitive chance that one's career trajectory might hinge on one's short-term performance, then that person will probably work very hard. These events may go on throughout a project's lifetime, and people will traverse through several up or down motivational phases during a project. Sometimes, the degree of staff motivation is obvious and can be modeled with relative certainty.

It is well known that people may use some time during the day to catch up on personal errands or do other nonproductive activities. Several studies have indicated that usually only about 60% of a person's time is actually spent producing (reference the exhaustion model preceding this section). This is related to Parkinson's Law, which holds that work expands to fill up the available time.

Motivation is clearly a decisive factor in successful companies. The following passages from [Yamamura 1999] are from the article *Process Improvement Satisfies Employees* that describes experiences in an early Boeing CMM Level 5 organization:

> An inspired employee's performance is a powerful tool for an organization. Motivated employees work with unbounded energy, create innovative solutions, and, most importantly, have great attitudes. They are always looking for better ways to do their job, and they feel comfortable volunteering for special assignments. In addition, management and the employees have a sense of trust.
>
> Satisfied employees lead to successful programs. People that are well trained, fit the job, have the right skills, are motivated, and execute their tasks with passion are the most likely to help a project succeed. Management can promote employee satisfaction with a project with good planning, a disciplined development environment, growth opportunities, special assignments, mentoring, and a focus on process improvements. Management should not overlook one of the most powerful tools they have—the highly satisfied and motivated employee.

There are various incentive schemes that companies can offer to improve motivation. These may be in the form of career planning and guidance, training and mentoring programs, an open environment for proactive improvements, financial bonuses, awards, recreational facilities and activities, extracurricular events, and so on.

A recent article by [Humphrey, Konrad 2005] discusses motivation and process improvement. It discusses people issues in a broader context than just developers and their capabilities and motivations. Many people in an organization affect the work. They show how the attitudes and concerns of people in the entire integrated community can help or hurt the work of developing, supporting, and enhancing software.

4.7.1.1 Overtime

Pressure to work overtime can greatly influence motivation and overall productivity. Studies have indicated that a small amount of aggressive scheduling will tend to motivate people to produce more, but highly unreasonable goals will have the opposite effect. The management challenge is to strike the right balance. The following example picks up from Chapter 2, where an overtime productivity function was formulated. We will elaborate the function for additional effects and organizational variants.

The overtime function in Chapter 2 represents the dynamics in a typical company, but in reality different classes of organizations exhibit varying productivity multiplier curves. Overtime rates, like other dynamic effects, often depend on the type of industry. Compensation is a primary motivation factor in most cases, and the degree of pioneering work is also of importance. People who have large financial incentives (such as the stock options in start-up companies) tend to work much longer hours. For example, people in an entrepreneurial start-up situation are going to be much more willing to put in overtime as opposed to those in a civil servant position. They are expected to do so and generally are motivated by the chance to make an impact and a windfall of money if the start-up is successful. Civil service employees have no such motivation and may, in fact, expect job security even if they do not consistently work hard.

Table 4.5 shows typical maximum workweeks for representative software development environments. These may also be adjusted higher depending on whether standard pay or extra overtime pay is given beyond 40 hours. Except for the civil service sector,

Table 4.5. Representative maximum workweeks

Industry Type	Typical Maximum Workweek (Hours)
Civil service	40
Fortune 500 aerospace/defense	50–60
Fortune 500 commercial software (e.g., telecommunications, enterprise software)	50–70
High-technology start-ups with stock options	80–100

these workweeks may become even longer during schedule crunch periods immediately preceding product delivery dates.

Figure 4.26 shows a representative family of curves for these extreme situations. Any organization modeling the overtime factor should calibrate as best as possible for their own environment and culture. Different organizational behaviors varying by industry or domain are seen for many phenomena. See the previous section for representative workweeks and later sections on attrition rates for selected industries.

The curves shown level off, and when extrapolated further into the oversaturated zone they eventually show degraded productivity as the expectation is set too high. For many individuals, the overtime productivity curve will exhibit a downward trend after a point, representing a retrenchment or purposeful backlash against (unreasonable) management policy. They may feel they have already paid their dues of heavy

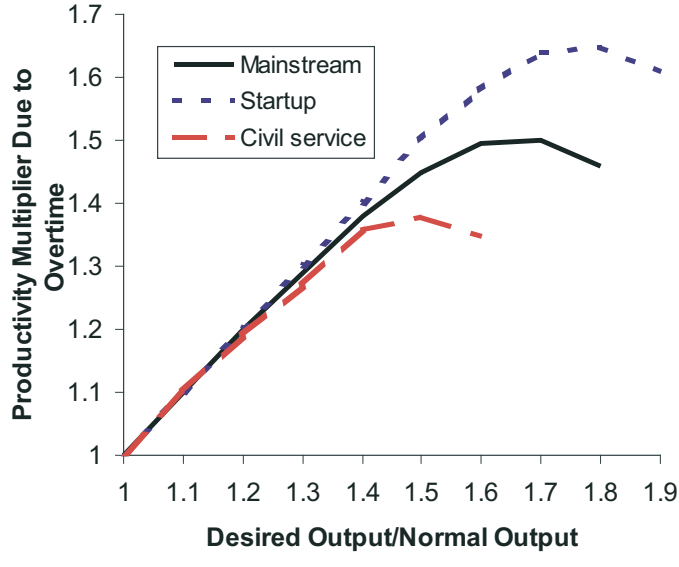

Figure 4.26. Family of curves for productivity due to overtime.

overtime and no matter what, they are going to take care of things other than work.

The deleterious effects of overly aggressive schedules are realized in other ways such as increased error generation (see the Abdel-Hamid project model for example) and personnel attrition. Too high pressure will cause more turnover, so the new learning curves and ramping up for the replacements have to be accounted for.

Mandatory overtime can also lead to the destruction of well-working teams as described in [DeMarco, Lister 1999]. Since not everyone can participate equally in overtime due to outside responsibilities, a divide widens on the team and it can eventually lead to total team breakdown. These are some of the ways that aggressive and unrealistic scheduling can be cross-purposeful. The indirect costs of overtime can easily outstrip any temporary gains in productivity. Another negative aspect of sustained overwork is modeled in the section on exhaustion and burnout.

4.7.1.2 Slack Time

The near opposite of overtime is slack time. The importance of having slack time is described in Tom DeMarco's book *Slack: Getting Past Burnout, Busywork, and the Myth of Total Efficiency* [DeMarco 2001]. Slack is necessary for people to get inspiration for new ideas and be creative. Running too tight a ship and requiring overtime works against that. Managers should foster environments where slack time is possible.

DeMarco also quantifies the effects of overtime in terms of probability of meeting schedule. Figure 4.27 shows that relationship from [DeMarco 2001]. The effect he is modeling is a schedule-oriented view of the overtime phenomenon modeled in Section 4.7.1.1. In region I, workers respond to pressure by trimming waste and concentrating on the critical path. In region II, they are getting tired and feeling added life pressure, so some undertime is taken. In region III, they have had enough and are looking for other work. This relationship may serve as reference behavior for a simulation project investigating slack time effects.

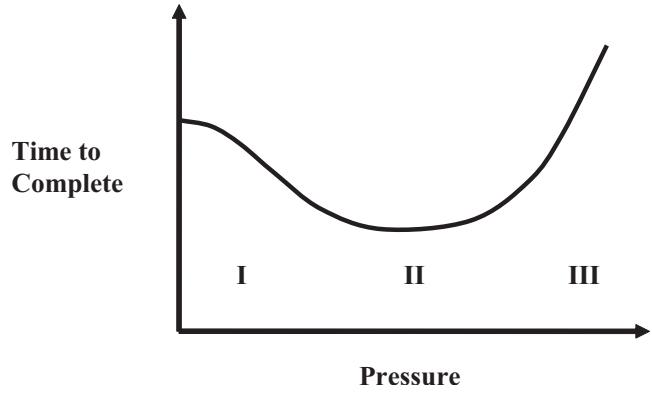

Figure 4.27. Effect of pressure on completion time, from [DeMarco 2001].

4.7.2 Personnel Hiring and Retention

Acquiring good personnel is often very difficult, but is time well spent. Personnel must first be found and identified. Then they have to be sold on a particular job, and negotiations ensue until all parties reach a hiring agreement. But acquisition is only part of the battle. People need to stick around long enough to recoup the investments made in them.

4.7.2.1 Hiring Delays

The time needed to hire new employees depends on the position levels, organizational culture and processes, current market trends, and so on. There are a number of steps in the overall hiring process at which things can slow down. The average hiring delay time for a position increases greatly with the level of experience and specialized knowledge sought. Table 4.6 shows some representative delay times for standard positions that can be used with a model for hiring delays.

The ranges in Table 4.6 account for differences in organizational practices, market conditions, and so on. A nimble, small startup company can accelerate hiring to a fraction of the time needed by a large entrenched organization requiring multiple approvals and background checks. It has been reported that Internet technology has been able to reduce the time for hiring from a 2–3 months to a few short weeks for some companies. The gains are presumably in the candidate location phase and not the resulting interviews, decision making, negotiations, and so on.

The ranges also account for some of the management controllables. For example, there are ways that the hiring process can be accelerated, like allocating enough money up-front for prime candidates to avoid offer iteration, hiring bonuses, or creative ways to lure good prospective employees. The hiring personnel, often human resources, can be given explicit priorities or be motivated otherwise to expedite hiring.

4.7.2.1.1 EXAMPLE: HIRING DELAY MODEL. A simple delay structure is a convenient way to model hiring delays. The delay time refers to the period from when a position is open, normally through a job requisition process, to when a new hire starts work. The hiring delay from model in Chapter 3, also shown in Figure 4.28, is used for sensitivity testing in Figure 4.29. For an entry-level person, the average hiring delay is varied from 1 to 3 months to gauge the effect on the hiring ramp-up. See the model, *hiring delay.itm.*

Table 4.6. Representative software development hiring delays

Position	Typical Hiring Delays
Entry-level programmer	1–3 months
Senior technical person/team lead	2–6 months
Software manager	3–8 months
Software director	6–12 months

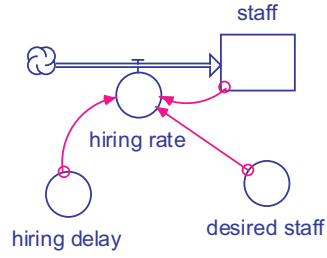

Figure 4.28. Hiring delay structure.

4.7.2.2 Attrition

Attrition should be minimized to reap the benefits of gained experience and to maintain a corporate memory. Employee turnover has been identified in recent years as an important factor in process performance. It is one of the newer factors in the COCOMO II model.

An ACM special interest group undertook a study on factors influencing employee turnover [Palmer et al. 1998]. This article has some good guidelines for those wrestling with retaining employees. It also lists some classic references about employee turnover issues for general industry. It is part of a conference proceedings on computer personnel that discusses several other highly relevant people issues.

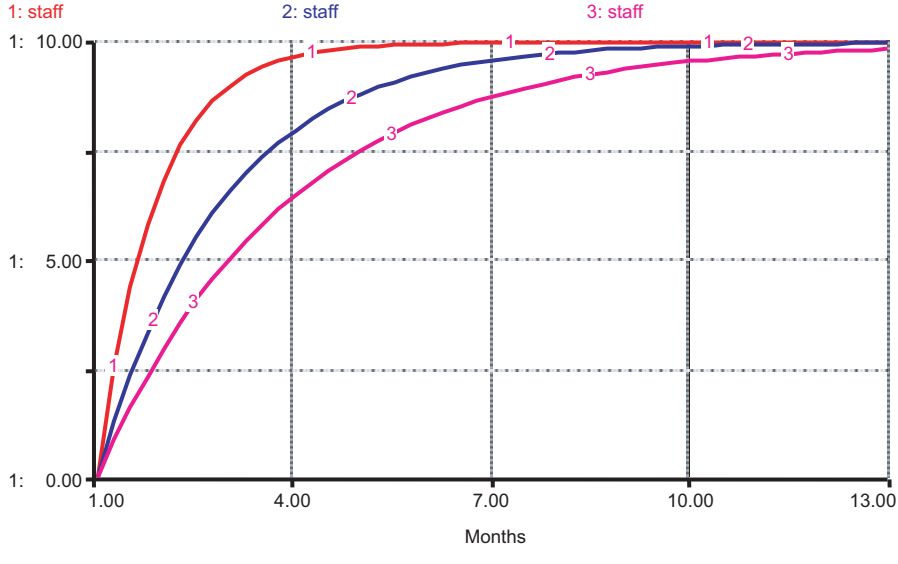

Figure 4.29. Hiring delay sensitivity (1: hiring delay = 1 month, 2: hiring delay = 2 months, 3: hiring delay = 3 months).

Table 4.7 shows typical personnel attrition rates for standard industry working environments. These are aggregate rates that include all ranges of experience. Younger personnel tend to move between jobs more often than senior people, so the rates listed may need adjustment depending on the respective levels of experience within an organization. During periods of business instability, the Fortune 500 companies may exhibit higher attrition by an additional 10%–15%.

Attrition rates are also localized geographically. Most notably, Silicon Valley and other pockets of high-technology start-up companies exhibit much higher attrition rates across all industry types due to the plethora of opportunities available.

These rates could change substantially given different economic conditions. We know that the dynamic pendulum swings ever quicker now; witness the droves of people who left for Internet firms in 1996–2000 and the migration back in the dot com shakeout shortly thereafter.

Some have argued that the type of process itself can lead to more stability in the workforce. Watts Humphrey has reported that the attrition rates observed when following the Team Software Process (TSP) are essentially zero [Humphrey 1999], though this claim should be followed up and analyzed with more samples.

In *Peopleware,* Demarco and Lister explain that there are both visible and not so visible signs of attrition. It is the not so visible signs that can have far greater effects. A visible sign would be manpower expense. A primary not so visible effect is the tendency of people to lean toward a short-term perspective, which becomes destructive.

4.7.2.3 Workforce Shortage

There is a seemingly ever-present shortage of skilled software and IT personnel to meet market needs. Closing the gap entails long-term solutions for university education, ongoing skill development in industry, and worker incentives. Howard Rubin has studied the IT workforce shortage in the United States [Rubin 1997]. One of the problem issues he mentions is a destructive feedback loop whereby labor shortages cause further deterioration in the market.

The degenerative feedback described by Rubin is a reinforcing (positive feedback) process between worker shortages, staff turnover, and job retention. A starting premise is that new people on a project are less productive at first, and require a drain on expe-

Table 4.7. Representative software development attrition rates

Industry Type	Typical Personnel Attrition Rates* (percent attrition per year)
Civil service	5%–10%
Fortune 500 aerospace/defense	5%–20%
Fortune 500 commercial software (e.g., telecommunications, enterprise software)	5%–25%
High-technology start-ups with stock options	20%–50%

*See the discussion above for modifiers due to business instability and geography.

rienced people needed to teach the new hires. The learning curve and training overhead are primary effects already described in the Brooks's Law model. This situation further causes an overall shortage due to the lessened productivity.

Organizations try to alleviate the labor shortage by offering salary increases to attract new hires. As more people job hop for better compensation, the shortage problem is exacerbated because more attrition and shorter job tenures cause lessened productivity. Thus, the market response adds fuel to the destructive feedback loop. The increased compensation to reduce the impact of attrition causes more attrition.

The feedback results in a greater need for people to develop a fixed amount of software, in addition to the overall increased demand for software development in the world market. Considering the larger world picture illustrates interactions between national and global economic trends.

Also see Chapter 7 on global and national workforce shortage issues. The report in [DoD 2000] includes shortages for particular types of software personnel categories, and provides recommendations for long-term solutions.

4.7.2.3.1 EXAMPLE: WORKFORCE SHORTAGE CAUSAL LOOP. Figure 4.30 is a causal loop structure of the labor shortage feedback dynamics. Desired productivity is that required to meet software demands in the market. The difference between desired productivity and actual productivity (on a local or aggregate level) manifests itself as a workforce shortage. The shortage causes aggressive hiring such as providing large starting salaries (or other hiring incentives) to fill the gap. The increased compensation available to workers causes higher attrition rates.

The increased personnel turnover causes learning overhead for new people to come up to speed and training overhead on the part of existing employees needed to train them. Both of these impact productivity negatively, which further exacerbates the workforce shortage. See the chapter exercises for further elaboration of labor shortage dynamics.

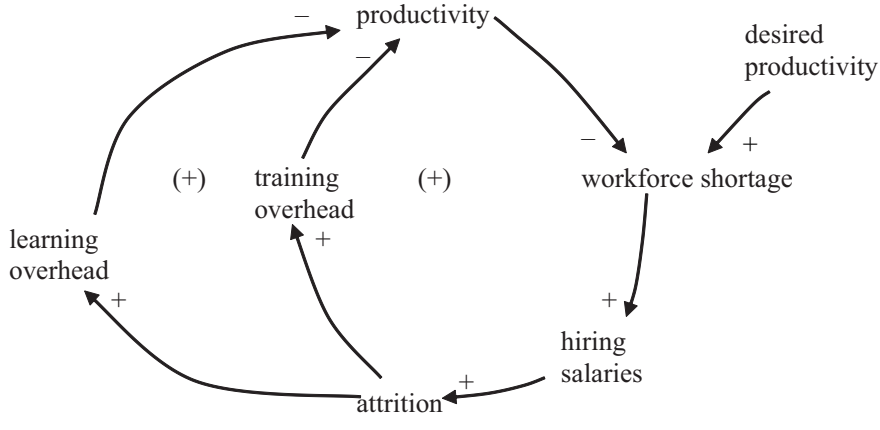

Figure 4.30. Workforce shortage feedback loop.

4.7.3 Skills and Capabilities

There have been very few models that account for specific skills other than aggregate experience and capability. Plekhanova performed research on integrating detailed software personnel skill profiles with simulation. A set-theoretic approach is used to describe the arrays of skills that people possess [Plekhanova 1999]. A number of capability and compatibility metrics were elaborated. With this information, resource-based process constraints scheduling can be determined based on critical human resource variables. Both individual and team characteristics impact the WBS, costs, time, and risks. Plekhanova presents a capability-based approach to stochastic software process modeling using a combination of discrete event simulation and system dynamics. The approach provides detailed process analyses, resource utilization, and how people resources fit a project not only at each discrete point in time, but also for each feedback path in the system.

Other process modeling work on people's competencies and characteristics for software development has been done by Acuña and colleagues. In [Acuña, Juzgado 2004], a capabilities-oriented process model is proposed that includes software process activities, products, techniques, and people, their roles, and capabilities. They define capability-person and capability-role relationships and procedures to determine capabilities and to assign people. The results of experiments confirm that assigning people to roles according to their capabilities improves software development.

A software process model handbook for incorporating people's capabilities [Acuña et al. 2005] covers people aspects. The focus is on extending software process definitions to more explicitly address people-related considerations. The authors developed a capabilities-oriented software process model formalized in UML and implemented in a tool. Also see [Acuña et al. 2006] on emphasizing human capabilities in software development.

4.7.4 Team Communication

Good communication between people is practically necessary for project success. Frequently, it is cited as the most important factor. Personnel capability factors (such as in COCOMO II) inherently account for team dynamics rather than assessing individual capabilities. If a team does not communicate well, then it does not matter how smart and capable the constituents are.

As previewed in the Brooks's Law model in Chapter 1, team size is an important determinant of communication and associated delays. Communication overhead depends on the number of people. The number of communication paths increases proportionally to the square of the number of people communicating. The Brooks's Law model shows that adding people increases communication overhead and may slow the project down. It takes much longer to disseminate information on large teams compared to small teams. This could have a profound impact on project startup, since the vision takes much longer to spread on a large team.

As the overall team size increases, however, team partitioning takes place and communication overhead is reduced on a local scale. Partitioning is not trivial to model and

the communication overhead should be reformulated to account for it. See Exercises for Chapters 2 and 6 for further discussion and elaboration of the Brooks's Law model for partitioning.

DeMarco and Lister discuss "teamicide" in *Peopleware,* and identify issues that affect team communication and formation. These include defensive management, bureaucracy, physical separation, fragmentation of people's time, quality reduction, phony deadlines, and clique control.

Additionally, motivational accessories may end up harming team communication. Accessories include award recognition, plaques, and so on. Bad feelings may emerge from competition and jealousy. Overtime can also affect the team chemistry; not everyone will be as willing or able to work longer hours. Other factors that may affect team communication include individual personality traits. Some team members may have dominating personalities, whereas others may be reluctant participants.

4.7.5 Negotiation and Collaboration

Achieving early agreements between project stakeholders is of paramount importance, otherwise it is likely that all stakeholders will not be satisfied and the project becomes a failure. Reaching a common vision between people and identifying their respective win conditions are major precepts of the WinWin process. Negotiation and collaboration processes can be difficult, though, and modeling the social aspects can provide insight. Several models have explored both the social and organizational issues of requirements development.

An early state model of the WinWin requirements negotiation was developed by Lee [Lee 1996]. It was a formal model of the infrastructure and dynamics of requirements collaboration and negotiation. This work established foundations for subsequent WinWin tools that help guide stakeholder involvement and collaboration, including the Easy WinWin process tool mentioned later in this section.

An important application in this area is [Christie, Staley 2000]* that simulates the organizational and social dynamics of a software requirements development process because they are so crucial to project success. The model explores organizational issues of requirements development as well as the social issues to see how the effectiveness of people interactions affects the resulting quality and timeliness of the output.

The model simulates a Joint Application Development (JAD) process [Wood, Silver 1995] that was implemented by the Computer Sciences Corporation for the U.S. Air Force. It was used to provide insights into the behavioral characteristics of a JAD requirements process in terms of resource needs and constraints, cycle times, quality, the effects of feedbacks on the system's dynamics, and the impact of social interactions on organizational dynamics.

JAD was used to identify detailed requirements for a large incremental release of a command and control system. Using the JAD process, system users and engineering specialists worked jointly in an electronic meeting environment and successfully re-

*This work probably would have continued had not Dr. Alan Christie, a highly respected researcher in software process simulation, passed away in 2002.

duced development time and failure risk. The outputs from the JAD sessions were then assessed by two independent reviewers and occasionally upper management became involved in decision making. Nine separate working groups were modeled; each group consisted of people in prescribed roles.

The model covers feedback between the social and organizational components resulting from the interpersonal effectiveness within the nine groups. The quality of the outcome from these groups is dependent on personal and social attributes, and this quality impacts the duration of the review process and how long subsequent corrective action will take. A Monte Carlo approach was used identify the quality value associated with a specified management threshold.

The model has both continuous and discrete modeling components, and a challenge was integrating the aspects. Although the modeling of the organizational processes uses a discrete, event-based approach, the "social" model was based on continuous simulation. The social model gives each individual three characteristics:

1. Technical capability
2. Ability to win over other team members to their point of view (influence)
3. Degree to which they are open to considering the ideas of others (openness).

An important conclusion from the work was that social simulation (i.e., the modeling of the interaction between individuals) needs to play a more important role in modeling software processes. The model showed that the level of commitment or trust escalates (or decreases) in the context of a software project. Commitment toward a course of action does not happen all of a sudden but builds over a period of time. Social phenomena can clearly be key in determining the outcomes of software processes and should be considered for realistic planning.

Another conclusion was that since organizational modeling focuses on "things" (e.g., people and artifacts), discrete simulation is appropriate. On the other hand, since social modeling more often focuses on parameters that vary smoothly, continuous simulation appears to be more appropriate. The Christie JAD model is a potential basis for modeling WinWin negotiations (see the chapter exercises).

Another application using system dynamics to model stakeholder collaboration activities is [Stallinger, Gruenbacher 2001]. They simulated the Easy WinWin requirements negotiation process to assess issues associated with the social and behavioral aspects, and to explore how the outcome is impacted. Easy WinWin attempts to foster stakeholder cooperation and involvement through tool guidance. They revisited the previous work by Christie and Staley and incorporated some of their concepts, including the modeling of an individual's characteristics for technical ability, ability to influence others, and openness to influence from others.

A method called Quantitative WinWin [Ruhe et al. 2003] uses simulation to support trade-off analysis for requirements selection. In this evolutionary approach to requirements negotiation, an analytic hierarchy process is used iteratively to balance stakeholder preferences with respect to classes of requirements. Next, a simulation model, GENSIM, is used to predict and rebalance the impact of effort, time, and quality. Al-

ternative solutions are offered and trade-offs made so that a small number of possible sets of requirements can be selected.

4.7.6 Simulation for Personnel Training

Simulation models are highly effective tools for training and education. Students and working professionals can interact with the simulations instead of learning the hard and expensive way on real projects. Simulation is used in many other fields to reduce risk, but has not been fully utilized in software engineering to date. For example, pilots spend thousands of hours in simulators before flying new planes to hone their decision-making skills. It is cheaper and much less risky compared to flying in a real plane the first time. Software people can also avoid some expensive mistakes on real projects by practicing first via simulation. The analogy to flying is so strong that sometimes the interactive training is referred to as "flight simulation."

Flight simulation is interactively running a model such that parameters can be adjusted midstream during execution. This mode lends itself to interactive decision making, whereby the user can influence the simulation outcome based on how the simulation has already proceeded. Flight simulation is an ideal training environment in which control decisions can be practiced risk-free.

Several efforts have used system dynamics for software process training in industry and academia. Deitmar Pfahl and colleagues have actively researched the usage of system dynamics modeling for training in [Pfahl 1995], [Pfahl et al. 2001] and [Pfahl et al. 2004a]. We have used simulation in a less formal and nonexperimental context at USC and at Litton Systems, as described in [Madachy, Tarbet 2000]. James Collofello at Arizona State University (ASU) has been using simulation for graduate software project management and industrial training [Collofello 2000]. A more general treatment of simulation for training can be found in [Drappa, Ludewig 2000].

A strong example of an empirical study using replicated experiments is [Pfahl et al. 2004b]. The experiments have shown that system-dynamics-based simulations are effective for understanding software project behavior patterns for project management and for learning multicausal thinking. An experimental group used a system dynamics model while the control group used COCOMO for project planning. Students using the simulations gained a better understanding of typical behavior patterns. Simulation models were found to be more advantageous than using static models like COCOMO. Students consistently rate the simulation-based role-play scenario very useful for learning about software project management issues. A recent overview of issues when using system dynamics for learning can be found in [Pfahl et al. 2006b]. After providing industrial examples, it discusses limitations, risks, and proposes future work.

The goal in [Collofello 2000] was to improve software management by having people practice with simulators. Favorable results were obtained using the simulator at Motorola that uses several models developed over the years at ASU. Some of the exercises included comparing life cycles and risk management approaches, and assessing the impact of software inspections, critical path scheduling, planning, and tracking. All participants found the simulator significantly added value to the course. The simula-

tions were also incorporated into a Web-based graduate-level software project management course.

Likewise, we have used system dynamics in the classroom at USC in both process modeling and general software engineering courses to teach concepts such as Brooks's Law, risk management approaches, and earned value management concepts. Many students were stimulated enough to take further classes and undertake research study in software process simulation.

We also had good success in industry at Litton Systems [Madachy, Tarbet 2000] when we introduced simulation in our training curriculum for software managers (see the next section). Some vendor tools have also incorporated system-dynamics-based training modules. Howard Rubin developed a software process flight simulation tool for the commercial market based on an implementation of the Abdel-Hamid integrated project model [Rubin et al. 1995].

Also refer to Chapter 7 for current and future directions of using simulation for training and game playing. The current direction is that interactive simulations are incorporating more intensive graphics and virtual reality to enhance the training experience. Chapter 7 describes how some other modeling techniques are doing this, and how interactive simulation-based training will be integrated with process mission control and analysis facilities.

4.7.6.1 Software Manager Training

Selected process management principles were successfully demonstrated through the use of live classroom simulations at Litton Systems [Madachy, Tarbet 2000]. Software managers and other designated leads received training from the Software Engineering Process Group (SEPG) in project management, software metrics, and other related subjects. Live simulations were used in the training venues for students to better visualize metrics trends and to improve their control techniques via interactive situations.

Use of the dynamic models enlivened the training sessions and stirred thought-provoking discussions. Using the models in real time allows for quick simulation runs to explore issues brought up during discussion. For example, a posed question may be translated into a model input and the simulation run for all to see the outcome. This often happens when presenting managerial subjects, as students propose specific scenarios that can be quickly evaluated through the simulation model.

The first models used for training purposes were an earned-value model, the Brooks's Law model, and a simple estimated productivity model. To teach earned-value techniques and project control, earned-value trends for a project were shown and trainees had to interpret them. Showing the different components of the earned-value algorithms helped them understand the method better. After interpreting the trends, corrective actions were suggested, as necessary. The simulation allowed some of the corrective actions to be modeled in order to see the impacts of their decisions. See Chapter 6 for more information on modeling earned value for training purposes.

The earned-value model at Litton Systems was also used as a basis for actual project control and evaluating the impact of requirements volatility (see Chapter 6). The Brooks's Law model was used to determine optimal staffing for some real projects.

The estimated productivity model helped to identify inconsistencies in some project plans. Using the same models for both training and real projects made the lessons stick much better than just leaving the models in the classroom.

Howard Rubin, Margaret Johnson, and Ed Yourdon also created a flight simulator to help managers understand the dynamics of new processes and technologies before they are introduced [Rubin et al. 1995]. The authors show how the tools can be used for information systems project management.

MAJOR REFERENCES

[Abdel-Hamid, Madnick 1991] Abdel-Hamid T. and Madnick S., *Software Project Dynamics,* Englewood Cliffs, NJ: Prentice-Hall, 1991.

[Acuña et al. 2005] Acuna S., Juristo N., Moreno A., and Mon A., *A Software Process Model Handbook for Incorporating People's Capabilities,* 2005.

[Curtis et al. 2001] Curtis B., Hefley B., and Miller S., *The People Capability Maturity Model,* Reading, MA: Addison Wesley, 2001.

[DeMarco, Lister 1999] DeMarco T. and Lister T., *Peopleware, Productive Projects and Teams,* Nes York: Dorset House Publishing, 1999.

[Humphrey 1997] Humphrey W., *Managing Technical People,* Reading, MA: Addison-Wesley, 1997.

[Raccoon 1996] Raccoon L., "A Learning Curve Primer for Software Engineers," *Software Engineering Notes,* ACM Sigsoft, January 1996.

[Weinberg 1998] Weinberg G., *The Psychology of Computer Programming: Silver Anniversary Edition,* New York: Dorset House Publishing, 1998.

4.9 CHAPTER 4 SUMMARY

People rather than technology provide the best chance to improve processes, increase productivity, and execute successful projects. They are the most important aspect to focus on overall because more can be gained by concentrating on people factors and human relations than technical methods. In this sense, agile approaches have it right. Processes are ultimately executed by people; hence, people can trump processes. The effectiveness of many processes and ultimate project success are dependent on people and their working together.

Motivation is always a key factor to achieve good productivity. It can enable people to work very hard and be creative. Yet people have limits on their amount of hard work and long hours. They can work at increased levels for only so long until burnout inevitably occurs. Then a period of de-exhausting is necessary.

Learning is an S-shaped phenomenon that can be successfully modeled, yet plans usually do not account for individual and team learning curves. The right environment must be in place for learning to occur because people are dependent on machines and usually other human beings for learning. Eventually, machine limitations will curtail the increased productivity due to learning.

Large projects undergoing change have needs for different kinds of people. We showed a simulation model that can be used to optimize the number of agile people on a hybrid process, in order to keep up with anticipated change traffic while still retaining pockets of stability for product dependability.

The right balance has to be struck in order to motivate people enough but not make them work so hard that a backlash occurs. In the end, too much overtime and pressure will negate any short-term gains. Management needs to be very careful about overtime and its converse, slack time.

Workforce issues should be dealt with at the macro level also. There are feedback dynamics that create and sustain a shortage of skilled people across the industry. Without a long-term vision to address the causes, the industry will continue to be plagued with staffing problems. Modeling and simulation can be used to help understand this and other people issues.

Companies should invest in retaining good employees. Personnel attrition can sink organizations. Making up for lost employees takes a lot of effort and entails hiring delays. Too much turnover causes everything to bog down, so whatever organizations can do to keep their (good) people around is worthwhile.

People also must have the appropriate skills at the right time. Those skills must first be defined and then the hard part of finding the right people or imparting new skills to present staff must occur.

Good communication between people is crucial, both within a development team and with external stakeholders. Individual capabilities are not as important as communicating well as a team. Emphasis should be placed on creating an environment for teams to gel.

Team dynamics play a particularly critical role when identifying and negotiating stakeholder win conditions. Teams generally will not work well together without open, honest communication and negotiation of win conditions to reach a shared vision between people. Theory W and WinWin negotiation methods have been shown to be useful for identifying and reconciling peoples' win conditions. For external stakeholders, expectations management is critical to make everyone a winner.

Process simulation is a valuable method for personnel training. It should be part of the training resource arsenal, like other methods of teaching and mentoring. Formal, replicated experiments have consistently shown that system dynamics is a viable and preferred method over static cost modeling in order to learn about complex project behavior patterns.

System dynamics is a good choice for modeling many people aspects. It is appropriate for those factors that tend to vary continuously (frequently, short-term fluctuations can reasonably be ignored) and/or can be considered in aggregate for the purpose of the analysis. The simplicity of system dynamics makes it handy to start modeling aspects that are difficult to model otherwise and thus ignored. Discrete approaches are better suited when it is prudent to model characteristics of individuals in a population and a continuous approximation is insufficient.

We have reviewed just some of the people phenomena that should be better understood and accounted for on software projects. Over time, more human aspects will be addressed by modeling and simulation to help promote people-oriented practices.

4.10 EXERCISES

These application exercises are potentially advanced and extensive projects.

4.1. Experiment with the Abdel-Hamid personnel sector under different staffing gap situations (inspired by [Johnson 1995]). Analyze the model behavior with three scenarios:

- The workforce is exactly the same the workforce level needed
- The workforce is less than the workforce level needed
- The workforce is greater than the workforce level needed

For the first case, set new hires to 0, experienced personnel to 12, and workforce level needed to 12. Run the simulation for 300 days. For the second scenario, make the gap large by setting new hires to 0, experienced personnel to 8, and workforce level needed to 12. Lastly, set new hires to 8, experienced personnel to 8, and workforce level needed to 12 to model a greater than needed workforce. Describe what happens in each scenario.

4.2. a. Research the literature for modeling human capabilities with system dynamics in other disciplines, or other simulation methods applied to software people. See [Acuña, Juzgado 2004] or [Acuña et al. 2005] for examples with software processes. Can any of the work be adapted for a system dynamics model for software processes?

b. Based on your research, develop new software process model(s) and carry out simulation experiments.

4.3. Identify some improvements to the Abdel-Hamid personnel sector model for your environment and implement them.

4.4. Develop and compare learning models for software development using both time and work volume as the independent variable. Does either seem most in line with intuition and experience? Why?

4.5. Perform some experiments to test the learning curve theories. Compare the model results to empirical data.

4.6. Extrapolate the overtime productivity curve into the "too high" region where productivity falls. Document the reasons why this happens and justify your values.

4.7. Model the interactions between overtime pressure and personnel attrition, and/or the interactions between overtime pressure and defect generation.

4.8. Read Tom Demarco's *Slack: Getting Past Burnout, Busywork, and the Myth of Total Efficiency* [DeMarco 2001] and create a simulation model for slack time. Integrate the phenomena he describes, some of which are already expressed as notional reference behaviors.

a. Identify and graph reference behavior(s) that the book describes about your topic.

b. Create a model, experiment with it, and write a report.

4.9. Read *Peopleware* [DeMarco, Lister 1999] to help define a project for modeling human factors. Some possibilities include Parkinson's Law, the office environment, team jelling dynamics, "teamicide," coding war games, overtime, corporate entropy, turnover, process improvement paradox, change dynamics, staffing dynamics, miscellaneous productivity factors, and more. Slack time concepts can also be included (see previous exercise).

a. Identify and graph reference behavior(s) that the book describes about your chosen topic.

b. Create a model, experiment with it, and write a report. Clearly identify what organizational "levers" are available to influence individual and group behavior.

4.10. Read *The Psychology of Computer Programming* [Weinberg 1998] to help define a project for modeling human factors.

a. Identify and graph reference behavior(s) that the book describes about your chosen topic.

b. Create a model, experiment with it, and write a report.

4.11. Evaluate the labor shortage degenerative feedback loop with an executable model.

4.12. Which effects in the IT workforce shortage causal loop diagram operate locally, globally (an aggregate of organizations or geographic provinces), or both? Define the system boundaries and enhance the diagram to model interactions between national shortages and global shortages. Develop a corresponding executable model.

4.13. Model collaboration dynamics between stakeholders. See [Christie, Staley 1999] or [Stallinger, Gruenbacher 2001] for some ideas or adapt their models. A review of the Easy WinWin tool used in [Stallinger, Gruenbacher 2001] or a similar groupware tool for negotiating requirements can provide insight on measures to include in the model.