19 Modeling and Simulation Toolset

Sergey Suslov
The AnyLogic Company

Dmitry Katalevsky
Russian Presidential Academy of National Economy and Public Administration (RANEPA)

CONTENTS

INTRODUCTION

This chapter is devoted to contemporary modeling and simulation techniques applied to complex project management. It is divided into two main parts:

- In the *first part*, we provide a theoretical overview of project management from the perspective of a systemic approach applying casual-loops diagrams, a methodology frequently used to produce a conceptual view of complex systems. We reflect typical causes of frequent project failures. We use a case study of a software development company to illustrate a common behavior when complex project spirals out of control. We provide a brief literature overview of project management simulations studies and give some practical advice for project management practitioners.
- In the *second part*, we provide an overview of three most widespread modeling techniques (system dynamics, agent-based, and discrete event modeling) using some

simple models of project management as examples. The second part of the chapter might be of interest to those who are interested to make their first steps in application of simulations to project modeling. AnyLogic software is used to illustrate case studies.

We provide overview of three the most useful simulation approaches – system dynamics (SD), discrete event simulation (DES) and agent-based modeling (ABM). While system dynamics is based on holistic process description, the ABM system uses a bottom-up approach that is described as interacting objects with their own behaviors. System behavior emerges as a summary of individual agents' actions. The main idea of DES is to consider the system as a process, that is, a sequence of operations being performed across entities. Depending on project' and organizational specifics, one can choose and apply the most useful simulation method case-by-case.

MODELING AND SIMULATION

Modeling plays a very significant role in our lives, yet people often underestimate how frequently they apply models. By simply calculating the time and cost of getting to the city center from the hotel while traveling, you are making use of a mental model. This model might even include an equation, for example, when you multiply the ticket price by the size of your family. Models can be observed everywhere in our daily life. They can come in a variety of forms, such as mental models, formulas, organizational charts, and physical models of buildings.

Modeling in a digital environment with the help of computer software tools has become a standard in business and engineering. Many vertical software solutions are used for complex project management in various industries. A good example of modeling in the construction industry is building information modeling (BIM). Currently, before any construction work can be started, a digital model of the building is created, using tools, for instance, Autodesk Revit. Such models include all the architectural and engineering aspects of the potential building (Figure 19.1 is an example of such a model).

It simplifies the construction project, because the technical side becomes transparent to all involved professionals (architects, structural engineers, construction professionals) and people who will use the building in future, such as industrial engineers that automate the factory when it is ready, or healthcare researchers who will organize processes in the future hospital.

Why do people create models? First of all, they might create a model because it is easier to experiment with a model than with real-life objects. Real-life experiments are often too expensive or even impossible to do, for example, the planning horizon for the construction of a new seaport will take tens of years. It is physically impossible to test which of the several processes in this port is best if we have several options; thus, a model makes a great alternative.

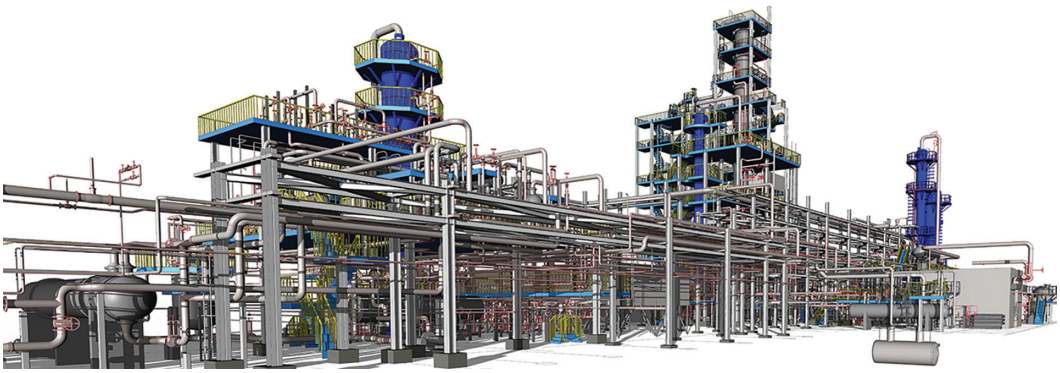

FIGURE 19.1 A building information model of oil refinery in Autodesk Revit software tool.

Unlike real-world projects, the virtual modeling world is completely risk-free. Managers, researchers, and engineers can create and test models of various system designs, and answer hundreds of what-if questions by experimenting virtually in a risk-free environment (Figure 19.2) (Borshchev 2013). Modeling is a very efficient instrument to improve people communication during the initial steps of a complex project, because it allows for objective discussion. There is no need to think that one person's opinion is better than the other's opinion because of differences in experience; managers and engineers can get modeling results to support their opinions. If we model the suggested scenarios, modeling will show which scenario has the lowest technical risks and the net present value (NPV) that fits our budget limitations.

In this chapter, we provide universal theoretical foundation of complex projects dynamics. Unlike real-world projects, the virtual modeling world is completely risk-free. Managers, researchers and engineers can create and test models of various system designs, and answer hundreds of what-if questions by experimenting virtually in a risk-free environment.

Simulation is unique among other modeling tools and technologies because it allows the user to add dynamics into models. The simulation model evolves over time, discretely or continuously changing its state. To create a simulation model is to define a set of rules that will drive the system over time.

Every simulation is a model, but not every model is a simulation. For example, the most widely used modeling tool is Microsoft Excel spreadsheets. Is a project's budget estimation in Excel a simulation? Definitely not, as it has no time component, it just includes slices of data on particular moments in time.

Simulation modeling is a very powerful instrument that is widely used for solving various business challenges. This is mostly because it is very practical and easy to understand, and there is no need to guess when you can just run a simulation model and see what happens. Animation is a significant advantage that simulation models can give, compared to static models such as those which are Excel-based. When people watch an animation like a video showing where a system, for example, a port is working with given equipment configuration and parameters, they find it easier to visually validate the model and trust its results.

Stochastics is another big advantage of simulation; it is possible to set up any characteristics of the model as a probability distribution with given parameters, let's say a project step duration. Each time the model is executed, a new random value is used as the parameter's value. If the model is executed many times, we can see the model result (e.g., total project completion time)

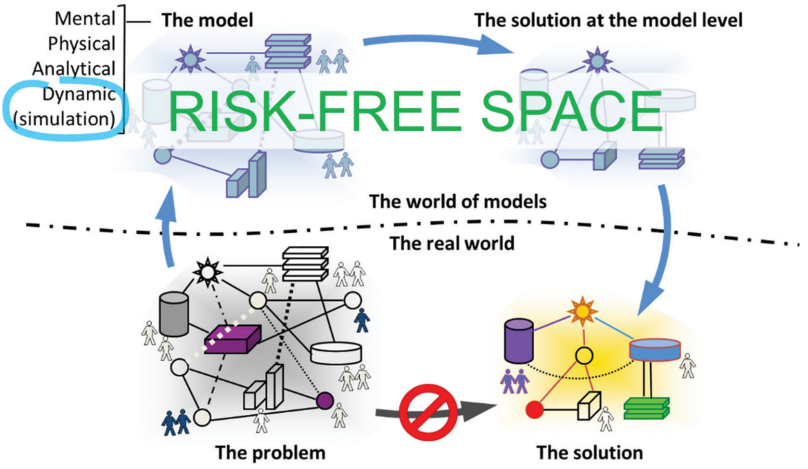

FIGURE 19.2 Risk-free virtual "world" of models.

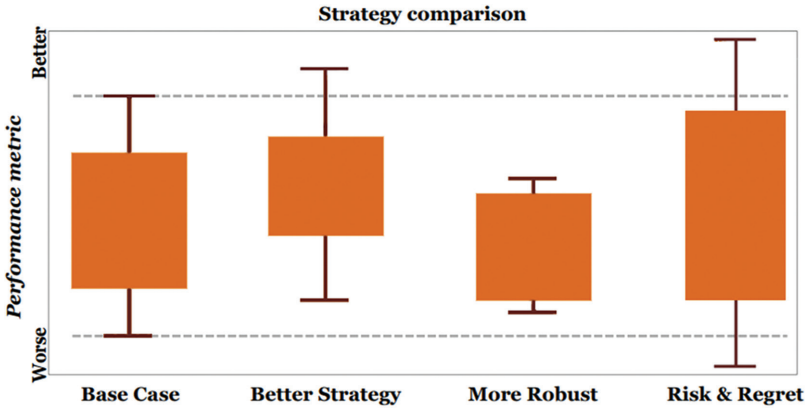

FIGURE 19.3 Box and whisker plot with simulation results of four alternative strategies.

as a stochastic value—create a histogram or box and whisker plot, and calculate mean and deviation. This allows you to take into account deviation, which represents the risks of being over project time or budget.

The box and whisker plot in Figure 19.3 shows the simulation results of four scenarios representing four different strategies. This plot allows you to compare them, taking into account not only average performance metrics (NPV, project duration, etc.) but also a deviation. "Box" represents a range with 50% results; "whisker" represents a whole range of simulation results (The AnyLogic Company 2016).

Time and stochasticity are the two key features which make the simulation model especially effective for use in complex project management. That is why we will continue to write about simulation only.

SIMULATIONS IN PROJECT MANAGEMENT

Simulation science has a *proven track record of studying complex projects*. For instance, the first research in simulation dates back to the 1960s, but only in the 1980s were the first project-specific modeling applications established. Since the 1990s, many publications have appeared, documenting usage of simulation approach in project management. Many studies of project management and simulations tried to grasp major reasons behind project failures and significant cost overruns being typical for this field of practice.

The first systematic studies of Roberts applied simulations (system dynamics, in particular) to project management by introducing flows of project work (job units) and concepts of perception gaps (differences between perceived progress and real progress). Together with underestimation of required scope and effort to complete a project, Roberts pointed out that these project errors were responsible for managements' distorted project perception, which in turn inevitably leads to resource misallocation (Roberts 1964, 1974). The first models produced by Roberts, Kelly referred to research and development projects and studied *perceived* versus *real progress* in projects as well as

Simulation science has a proven track record of studying complex projects. Practitioners of simulations believe that complex feedback structures around managerial perception of projects and managers' decision-making are the elements key to project disruption. Since projects (especially complex) are usually tightly coupled systems, it becomes increasingly important to assess potential ripple effects. Due to multiple interactions of non-linear feedback loops with unintended and counterintuitive consequences and their compound nature, complex projects often spiral out of control.

development of R&D dynamics over time and effects of multi-project management. These were the early attempts to investigate the impact of **managerial decisions** on project execution, based on the assumption that *perception of the project's state may be different from reality.*

Later on, Richardson and Pugh added the concept of Rework, undiscovered Rework, perceived progress, and real productivity (currently the classics of project management models in system dynamics). The model introduced by Richardson and Pugh (1981) concentrated on studying several key domains typical for any complex projects (monitoring and control cycle, Rework generation, and staff hiring).

Based on the achieved progress in project simulations, Pugh-Roberts Associates created the "Program Management Modeling System" (a set of complex system dynamics models used as a project management tool to support managerial decision-making). The tool was successfully applied in a number of management consulting projects for large construction and even dispute resolutions (delay and disruption (D&D) cases). One of the early successes was the Ingalls Shipbuilding case against the U.S. Navy in the 1970s. Ingalls Shipbuilding won a contract to build a number of warships for the Navy in 1969–1970. The contract price was fixed. The project resulted in about USD 500m cost overrun; however, the Navy agreed to reimburse only USD 150m direct costs, blaming the rest of the overruns for Ingalls' mismanagement. Ingalls sued the Navy, claiming that constant design changes caused the D&D. Pugh-Roberts Associates created a complex model, replicating in detail a shipbuilding project to quantify the cost of disruption resulting from delays and design changes from the customer. The case was settled out of court for USD 447m. In an investigation of the case, Cooper identified that about USD 200m–300m of the settlement could be attributed to the model-produced analysis (Cooper 1980).

Many researchers have made other important contributions to the field of simulations in project management over the years, to mention a few:

- Abdel-Hamid (1993) (integrative model of a software management project): explicit incorporation of managerial functions of planning and staffing linked to the process of software development.
- Cooper and PA Consulting (Cooper 1980, 1993a,b,c; Cooper et al. 2002; Cooper and Reichelt 2004): project management as a complex system, non-linear feedback, and quantification of ripple effects on cost overruns.
- Williams et al. (1995): the effects of design changes and delays on project costs (compounding effects).
- Ford and Sterman (1998): multiple phase project model (aging chain structure) explicitly portraying iteration over four distinct development activities to describe average development processes.
- Williams (1999): investigation into what constitutes complexity in project management.
- Graham (2000): insights from system dynamics modeling of complex programs.
- Lyneis et al. (2001): strategic management of complex projects.
- Eden et al. (2005): claim analysis of complex project failures (comparison of system dynamics modeling and "measured mile" methods).
- Taylor and Ford (2006): strategies for managing projects near tipping point.
- For example, Lee et al. (2007) investigated the interaction of resource allocation delays and different amounts of control imposed by managers and made some counter-intuitive conclusions (i.e., because delays are inevitable, optimal delays with minimum timing are preferable) and many more.

As a result of cumulative studies of project management failures, the simulation practitioners have developed a unified theory of a *typical project management disruption mechanism*. We summarize it below using a case study of a software developing company to illustrate a step-by-step typical project management disruption dynamics.

A CASE STUDY: A MECHANISM OF A PROJECT DISRUPTION

There is an important difference between traditional approaches to project management and simulation modeling (i.e., system dynamics, agent-based modeling (ABM), etc.). In traditional project management tools, critical path method (CPM)-based tools describe a project as a networked sequence of discrete technical tasks and events, and portray a project as the sum of discrete work segments. Such tools and systems can be badly misleading by failing to portray that projects *really do not work* in a straight line of tasks started and ended but in an iterative process of accomplishment. Further, they encourage the view of projects as projectiles, hurtling toward an outcome on which human intervention has little effect (Cooper 1994).

Practitioners of simulations and, in particular, of system dynamics widely believe that complex feedback structures around managerial perception of projects and managers' decision-making are the elements key to project disruption. It's usually our decisions and actions that work through the multiple non-linear cause-and-effect relationships constantly involving delays that affect project execution. In this section, we will review the typical mechanism of project disruption and propose several remedies and recommendations to mitigate negative effects and prevent project failures.

Let's imagine that ABC Company is trying to create a complex software project. We assume that in order to launch the software successfully, ABC Company staff needs to develop about 100 thousand lines of source code.[1] Using typical system dynamics language in VenSim software tool,[2] we can propose a simple structure (Figure 19.4).

Stock **Work to be Done** represents an initial projected work (i.e., 100 thousand code lines) that the ABC management thinks they need to implement to finish the software. The stock **Work Really Done** represents a finished piece of work that does not need any further work. Both **Work to be Done** and **Work Really Done** are measured in the number of program code lines. As the software development progresses, the **Work to be Done** is depleted and **Work Really Done** increases. **Work Being Done** is a flow of work at any given time over the software development period. It can be measured in code lines added per time period, that is, the number of lines of code added per day.

Typically, **People** and **Productivity** directly influence the speed of **Work Being Done** (Figure 19.5). One would assume that usually the more technical staff (software developers) are hired to work on the software and that the more productive they are at adding code per time period, the quicker the stock **Work Really Done** is growing. However, it is reasonable to assume that not all **Work Being Done** is useful and meets our quality criteria. The variable **Quality** should be included into our system, measuring a fraction of **Work Being Done** (between 0 and 1) that is actually entering the pool of **Work Really Done** and does not require any further work. For simplicity reasons, we may assume that **Quality** is linked to the number of generated errors in program code. In reality, there are many factors both internal and external that directly and indirectly influence Quality and Productivity.

As noted previously, a huge advancement in understanding project management dynamics was achieved by adding the concept of a **Rework Cycle** (see Figure 19.7). Errors are detected via the testing process. As errors are detected (depending on the error discovery rate), the Rework is identified, increasing the amount of work remaining. Typically, system dynamics practitioners use the stock **Known Rework** to mark the amount of work that needs to be redone. There is also a notion of **Undiscovered Rework**—the work that contains existing but undiscovered errors. For the purpose of simplification in our chapter, we will operate with the only **Rework** stock in this case. **Rework** is added by **Rework Discovery** flow (in our case, a level of code lines classified as needing a rewrite per time) and depleted by **Rework Done** flow (a level of code lines fixed per time period). **Rework Discovery** can be tricky. Sometimes Rework is quickly discovered; however, this might not

FIGURE 19.4 Main stocks and a flow in the model.

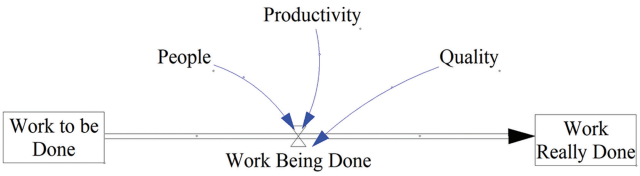

FIGURE 19.5 Influencing variables.

be always the case. The NASA study of flight software complexity shows (NASA Office of Chief Engineer 2009: 47) that many defects are inserted during the design and coding stages of the software development process (see Figure 19.6 for illustration). As Holzmann claims,

> The amount of control software needed to, say, fly a space mission is rapidly approaching a million lines of code. If we go by industry statistics, a really good—albeit expensive—development process can reduce the number of flaws in such code to somewhere in the order of 0.1 residual defects per 1,000 lines. (A residual defect is one that shows up after the code has been fully tested and delivered. The larger total number of defects hiding in the code is often referred to as the latent defects.) Thus, a system with one million lines of code should be expected to experience at least 100 defects while in operation.
>
> **Holzmann (2007)**

However, even **Rework Done** flow might contain errors that will result in another cycle of the Rework.

As Rodrigues and Bowers observe, the Rework cycle identifies four key factors that are partially under management control and should be treated with care: resource level, productivity, quality, and error discovery time (Rodrigues and Bowers 1996). Usually, project managers focus on resources and productivity—in ABC Company's case, people and their programming skills—treating them as fundamental prerequisites for project success or failure. However, the system dynamics experience suggests that it is equally important to monitor both quality and *error discovery rate*.

Lyneis and Ford consider the Rework cycle to be the most important feature of project disruption:

> The rework cycle is, in our opinion, the single most important feature of system dynamics project models. The rework cycle's recurrent nature in which rework generates more rework that generates more rework, etc., creates problematic behaviors that often stretch out over most of a project's duration and are the source of many project management challenges.
>
> *Lyneis and Ford (2007)*

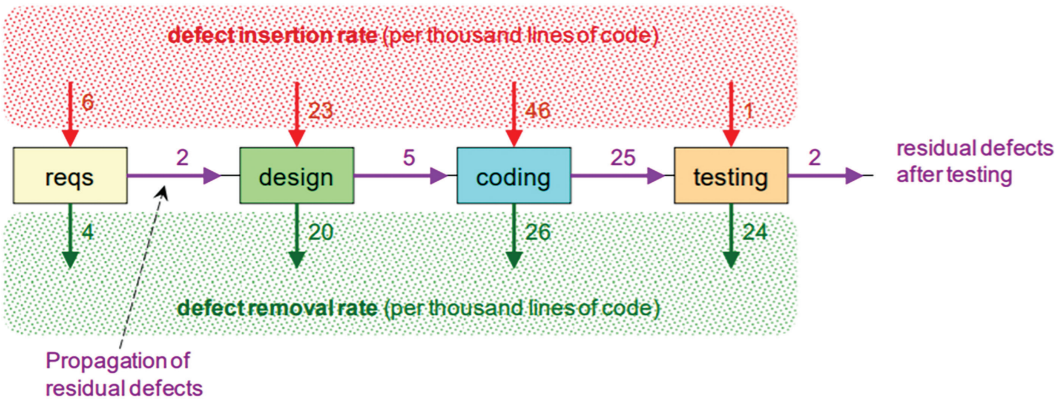

FIGURE 19.6 Defects propagation through various stages of software development (after testing, two defects remain per 1,000 lines of code) (Eick et al. 1992).

Rework cycle can easily be accountable for over 50% of the total time for project development (Cooper 1994) (Figure 19.7).

Further development of the ABC software project development case will add several cause–effect chains and feedback loops describing project lifecycle dynamics.

The management of ABC tries to control the project by evaluating its progress periodically. Primarily, they assess the necessity of getting additional personnel to finish the remaining work (including discovered Rework) on time. **Perceived Progress** translates into **Expected Completion Time**. If **Expected Completion Time** significantly deviates from the **Scheduled Completion Time**, the management needs to take action.

The most common approach reflecting conventional wisdom would be *to add more personnel* to increase the rate of **Work Being Done** (the cycle *Undiscovered Rework* → *Perceived Progress* → *Expected Completion Time* → *Scheduled Completion Time* → *Time Remaining* → *Staff needed* → *Staff Requested* → *Hiring* → *Staff on Project* in Figure 19.8).

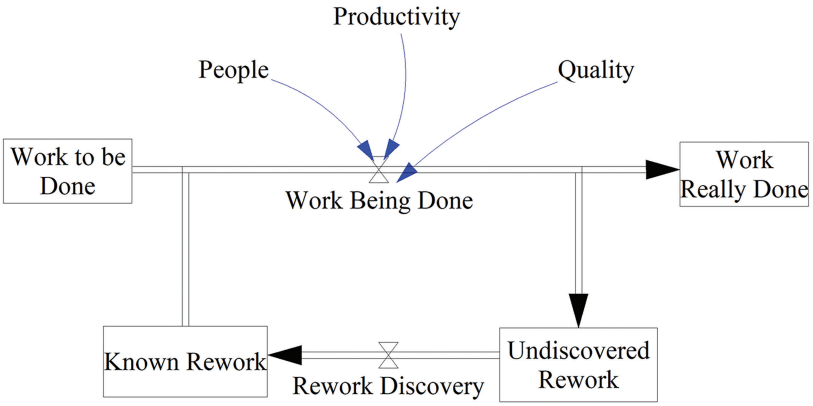

FIGURE 19.7 Typical Rework cycle.

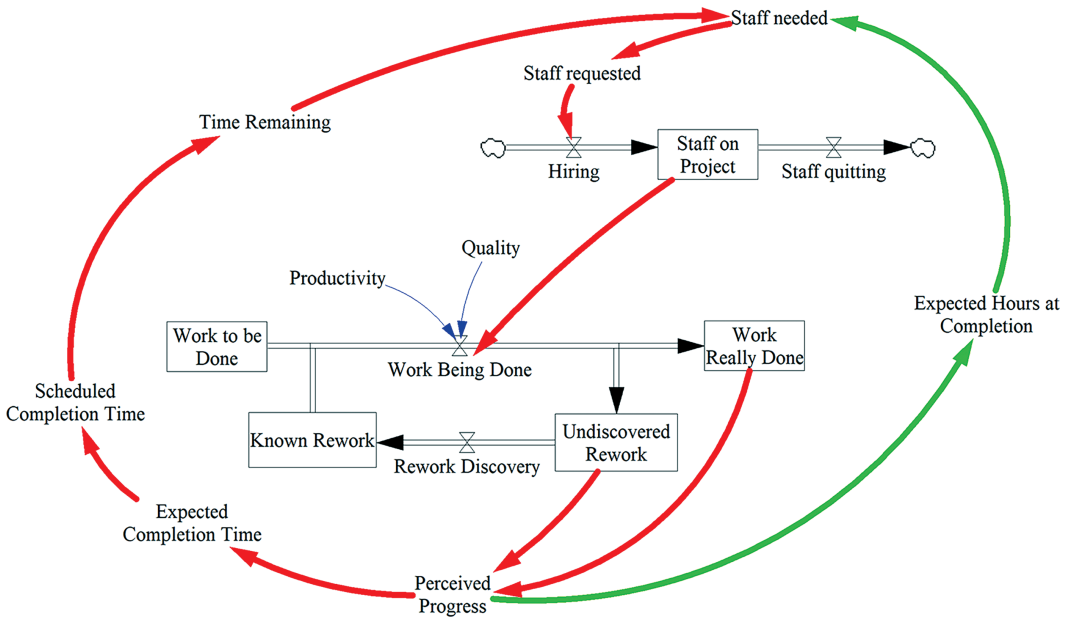

FIGURE 19.8 "We-need-more-staff!" cycle.

If this is not an option—for instance, due to resource constraints or unavailability of candidates with relevant qualifications and skills—the management will respond with a requirement of **extended work hours** for their software development team (overtime). This is the most common option to avoid the additional cost and commitment of bringing in additional people.

Overtime usage quickly becomes the new "normal" for the Company employees. The perceived gap between **Work Really Done** and **Work to be Done** still remains. Constant overtime with some delay effect reduces productivity and increases error rate (fatigue effect). More errors push the Rework cycle further, in turn increasing the amount of work that needs to be done and delaying work completion. The full cycle repeats itself. The **Overtime** feedback loop quickly becomes *self-reinforcing* (Figure 19.9a). Usually, the Overtime cycle and its

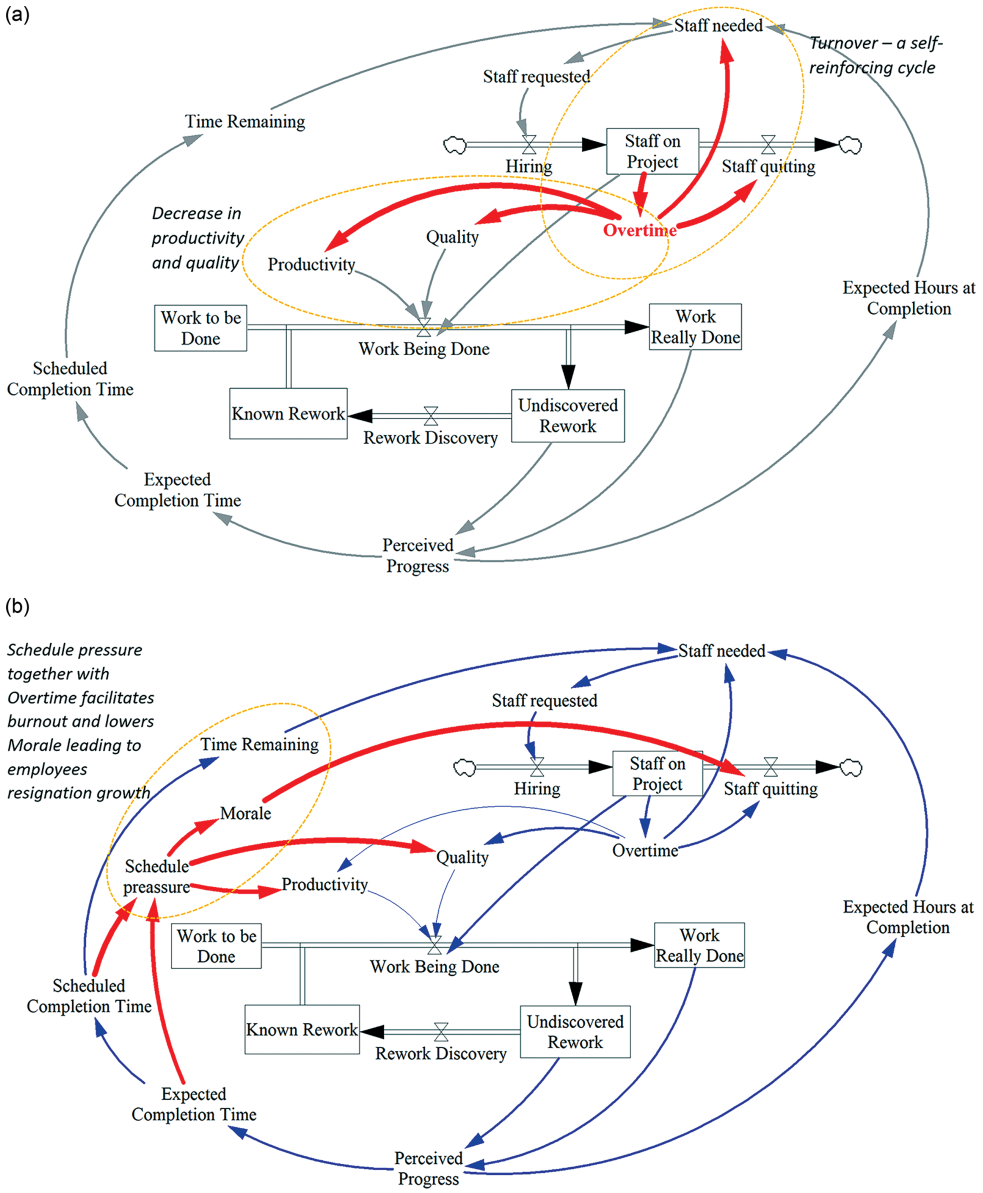

FIGURE 19.9 (a) Overtime cycle. (b) Schedule pressure and deteriorating morale added to the turnover cycle.

impact on fatigue and eroded personnel morale are significantly underestimated by management (Chan 2011; Neves et al. 2016).

After a few weeks, several software developers (the key staff, in fact) decide to quit ABC Company due to constant **Schedule Pressure** and deteriorating **Morale** (Figure 19.9b). Often, the highly qualified personnel leave first, since it's easier for them to find better jobs due to their qualifications and credentials. In addition to the fact that the project is experiencing a brain drain, the abandoned **Work to be Done** is redistributed on the personnel left further, exacerbating their performance and further pushing the self-reinforcing feedback loop of **Overtime**.

Now, management is committed to hiring or transferring additional personnel to the software project. However, it takes time to find and hire the right people (it is a process with a delay). Therefore, the overloaded remaining personnel further experience deteriorating morale and decreasing **Quality**.

However, bringing in new personnel has another counter-intuitive effect[3] which is usually underestimated or ignored. As new hires or specialists are transferred to supplement the current software team progress, new people with less experience or skill than those already on board enter the project. This is especially true if skills required are rare or in high demand on the labor market. The more constrained the labor market is, the lower the level of newcomers joining the project (Figure 19.10a). Therefore, it takes time for the newcomers to get themselves familiar with the project and go along the learning curve that the remaining staff has already been through.

There is a complex interaction of the feedback loops arising from the newcomers. Newly hired individuals might, intentionally or not, contribute to the project team turmoil by being in some cases less loyal, demonstrating higher attrition rates, being less skillful than expected (oversold), having unrealistic project expectations, etc. This is another vicious (self-reinforcing) cycle of *new hiring–higher departure rates–more hiring*.

New staffs require more supervision, and precise task-setting and control. Therefore, the experienced personnel's attention is now more and more diverted from the software development tasks to interacting with the newcomers, training, mentoring, and supervising them (Figure 19.10b). The management of ABC Company realizes that the software development process is now even more prone to the coding errors which increase the Rework cycle. The counterintuitive result of a substantial team expansion policy is *skill dilution*. Skill dilution has direct consequences such as lower productivity and further deteriorating quality (Figure 19.10a,b). The **Work Being Done**, Quality and Productivity are decreasing, while error rate, **Rework Discovery**, and **Rework are** increasing; the gap between **Work Really Done** and **Perceived Progress** is widening, which in fact suggests *to add even more project personnel* (!).

The management team cannot keep up with rising project costs. The growing organization becomes dysfunctional. It generates many meetings and discussions, while the responsibility is blurred. Another paradox is that despite the growing number of the project team, the availability of each person may even reduce, since personnel becomes sub-divided into smaller groups and much more time is now taken for internal discussions and reviews of design changes. As a result, the project team feels increasingly mentally and physically exhausted.

This contributes to burnout, lower morale, and all kind of negative consequences associated with it (Figure 19.11).

It can still get worse. Problems that occur in the early stages of the project quickly propagate to downstream work—poor requirements (unclear, incomplete, too general), software tests/evaluations lacking rigor and breadth, etc. can trigger significant Rework further (NASA Office of Chief Engineer 2009: 57). Classical examples include large construction, manufacturing, new product design, and other projects where changes in the project design and engineering stages affect construction and manufacturing stages.

It is possible that errors are discovered months and years later—a typical case in large and complex projects (construction, infrastructure, software, etc.). Some errors discovered may have

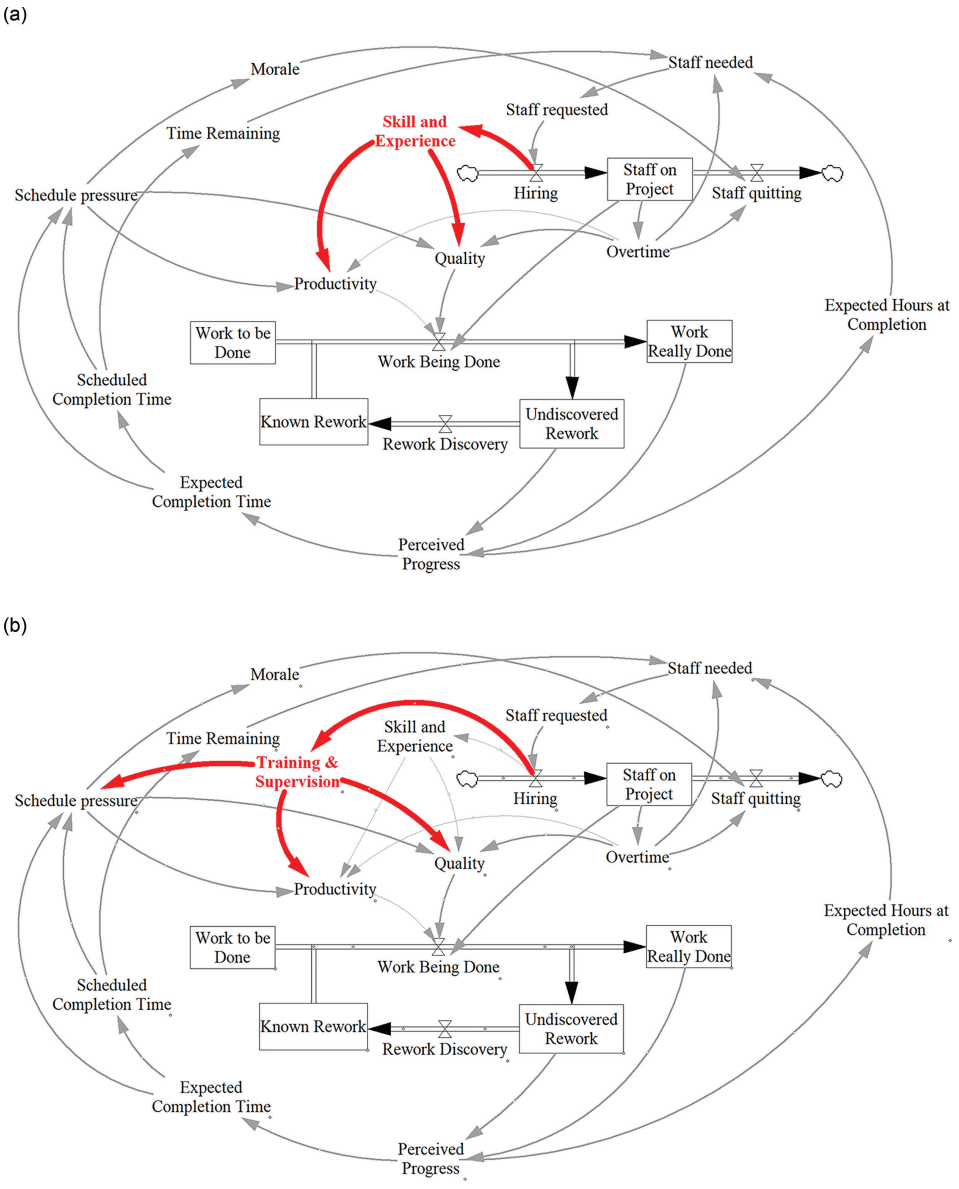

FIGURE 19.10 (a) Skill dilution from the new hires. (b) Training and supervision of newcomers diverts time from the project adding to schedule pressure.

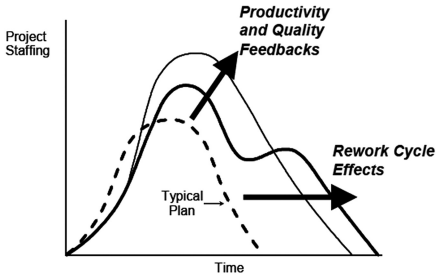

FIGURE 19.11 Typical rework cycle and productivity/quality effects on project staffing dynamics.[4]

devastating impacts on the project performance, necessitating changes that eliminate a substantial part of the project progress.

SOURCES OF RISKS FOR A TYPICAL PROJECT EXECUTION

Analysis of complex projects suggests that key sources of risks in terms of impact on delivery deadline and costs are as following:

- Late information or changes in initial design
- Constraints in resource availability (i.e., slow project start, insufficient skill mix, forced cost cuts to meet financial constraints, etc.)
- New processes, materials, and team members
- Management and organizational changes
- Initially aggressive project assumptions (compressed timing, inadequate budget, misperception of overall project complexity).

The degree of cost escalation found in software development processes was addressed by a number of researchers, including Boehm (1981), Larson and Wertz (1993) and Stecklein et al. (2004). Larson and Wertz (1993) conducted a study on satellite costs, estimating error costs over each lifecycle phase of the satellite production and developed the guidelines for NASA on estimation of the impact of errors. The NASA study (2004) distinguishes four main phases (requirement preparation, design, code, and testing phases) in the software development project and forecasts that the overall error cost factor would increase dramatically from phase to phase. Let's assume the cost of an error in the requirement preparation phase is 1 unit of cost. Then, the cost of fixing that error in the design phase increases to 5–7 units; in the coding phase, 10– 25 times; and in testing phase, more than 50 times (the NASA study compared different methods to calculate escalation of costs and revealed that cost growth rises exponentially over the project lifecycle). Software projects as well as many large-scale complex projects are prone to so-called "killer errors".

Since projects (especially complex) are usually tightly coupled systems, it becomes increasingly important to assess potential ripple effects. Due to multiple interactions of non-linear feedback loops with unintended and counterintuitive consequences and their compound nature, complex projects often spiral out of control.

Even insignificant by itself, residual mistakes in software development or small independent failures in a complex project combined can cause a major disruption. NASA's expert in complex software development G. Holzmann concludes that adding backup and fault protection translates into increase in a system's size and complexity with "unplanned couplings between otherwise independent system components.…. Given the magnitude of the number of possible failure combinations, there simply isn't enough time to address them all in a systematic software testing process. For example, just hundred residual defects might occur in close to ten thousand different combinations" (Holzmann 2007). A combination of minor failures was directly responsible for the failure of NASA's Mars Global Surveyor mission (Holzmann 2009).

Conclusion. Complex projects are prone to errors and disruptions. Humans are typically not very good at working with tightly coupled systems while the cost of error sometimes is too high (i.e., human life). Ability to predict various scenarios of the future development of the project and to quantify potential outcomes and impact of managerial decisions on project outcomes becomes a critical skill. Therefore, simulations start to play an increasingly important role.

In the next part of the chapter, we will provide some insight into simulation science reviewing three key approaches to project simulations coupled with simple models and will discuss advantages of each approach.

OVERVIEW OF KEY MODELING TECHNIQUES

The simulation technics applicable to complex project management is dominated by three paradigms of thinking: system dynamics, discrete event, and Agent-based modeling (ABM). We should say that we see applications of the same three modeling approaches across business system simulations. Let's go into details and see what these approaches are.

System Dynamics

System dynamics is the oldest simulation approach, with roots in work done by MIT Professor J. Forrester in 1950. When joining MIT Sloan School of Management in 1956, J. Forrester was already famous for his work for the U.S. military on the first U.S. air defense system (Semi-Automatic Ground Environment, SAGE), random access memory (RAM) for computer industry, computer numerical control (CNC) machines for manufacturing. Forrester applied this significant experience in technical system design to business and social simulations.

Initially, J. Forrester called the new approach industrial dynamics. He later reviewed its name to system dynamics to highlight that economics consists of complex systems with non-linear non-obvious behaviors which are interconnected in dynamics.

From the very beginning, system dynamics stayed focused on the management and organizational layer, abstracting from the engineering and manufacturing sides of the business. This was established because of the idea that most problems are caused by the management and organizational side of the business.

Let's try to build a model of a software development project, which is a good example of a complex project involving a lot of people. System dynamics is supported by three to four tools: VenSim, PowerSim, iThink, and AnyLogic that use one and the same notation for model building. We will learn its basic element—stock and flow diagram in this exercise.

A stock is a digital representation of something that is modeled, for example, a number of people, a set of requirements, and a number of products. Any process in system dynamics is modeled as a flow between stocks. In our example, software development is a flow between *requirements* and *developed software* (Figure 19.12). You can imagine that any production process is just a raw material flow to finished goods.

This is already the simplest model that will work if we know the *software development rate*, but it is not a constant; it depends on the number of *staff*, their *productivity*, and even their *communication overhead*. Adding such variables to our model makes it a bit more complicated (Figure 19.13).

The model is graphical and easy to understand, but it cannot be executed until we add numbers and formulas to connected variable flows.

Productivity is just a constant; *staff* is a variable that we can set up by a value at this stage. You may see that stocks, flows, variables, and constants are graphically represented differently. It is a standard system dynamics notation supported by a variety of tools. The screenshots you see in our chapter were captured from AnyLogic software.[5] We will continue illustrating our model

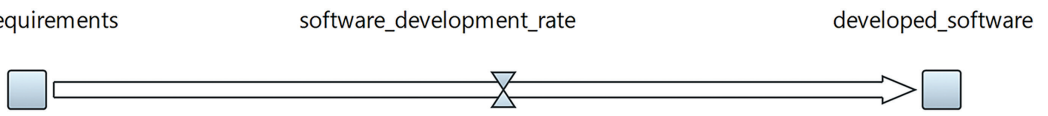

FIGURE 19.12 Simplest system dynamics diagram of the software development project.

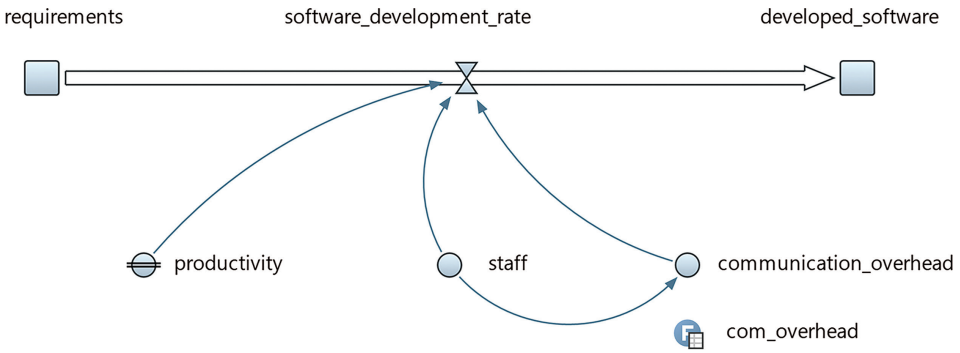

FIGURE 19.13 Modified diagram of the same model.

development process with screenshots from the AnyLogic software, since it supports all major simulation approaches. Some elements may be new to you, but never mind, we will explain all significant elements of the model.

We will define *communication overhead* with a table function that is also typical of system dynamics models. Such table functions can be a result of real system observations. Table function includes pairs of arguments and values. It allows you to set up a function when you do not know a formula, the function will be defined and data approximated.

The horizontal axis on Figure 19.14 represents team size, and the vertical axis shows overhead percentage. When we have less than five people, we have no overhead on their communication; but if we start with five and later on the number grows, the more people are working on the project, the bigger overhead they generate.

In AnyLogic, we call functions by function names with parameters in parentheses. We will define *communication overhead* as it is shown as follows:

$$communication\_overhead = com\_overhead(staff)$$

The software development rate is equal to productivity multiplied by the number of staff and the communication overhead.

$$software\_development\_rate = productivity * \left(1 - \frac{communication\_overhead}{100}\right) * staff$$

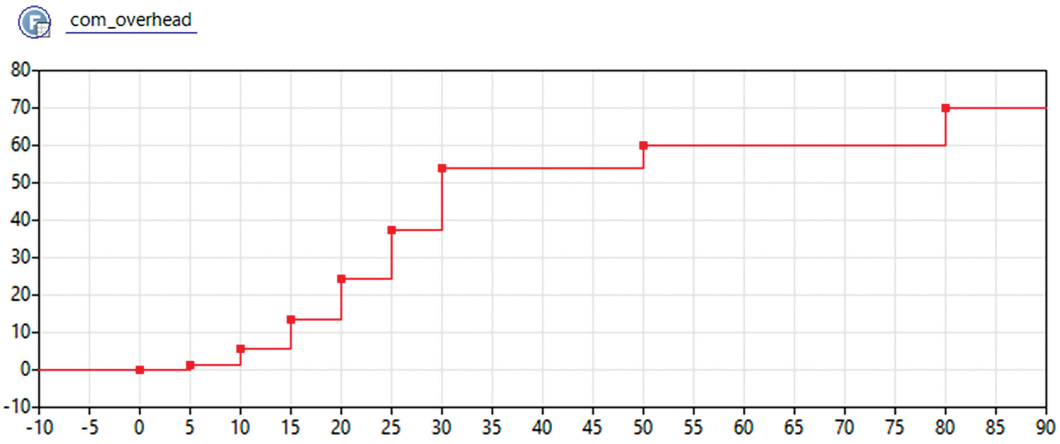

FIGURE 19.14 A table function that defines communication overhead.

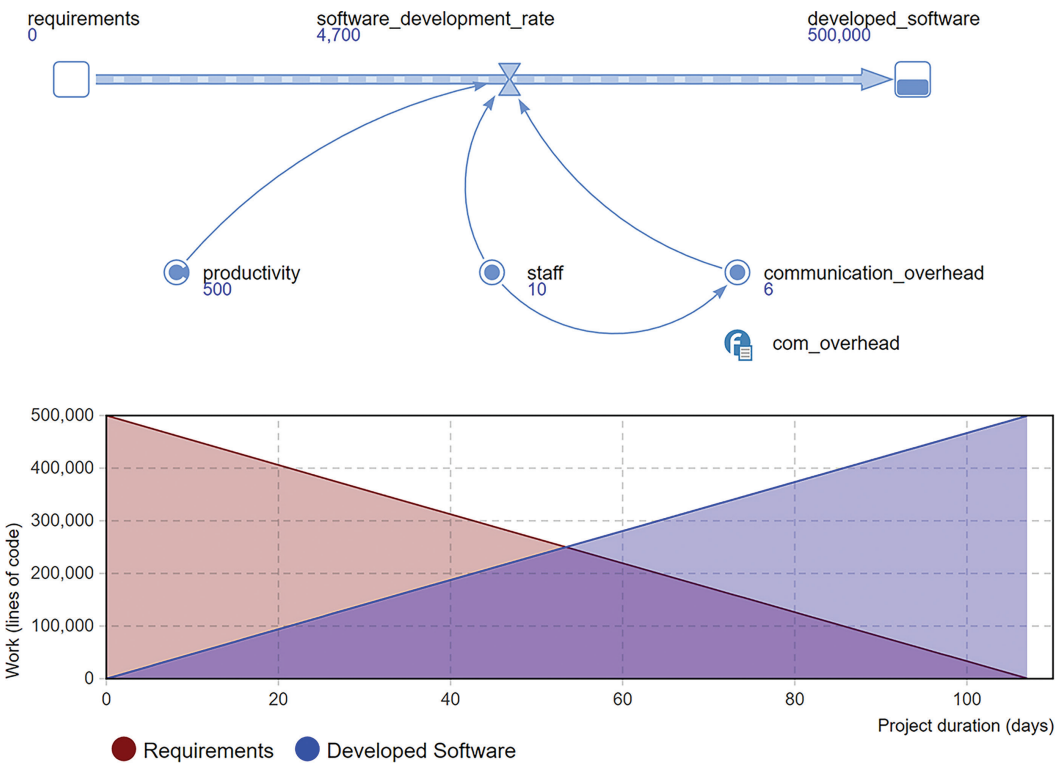

FIGURE 19.15 Modeling results, ten people, 500,000 lines of code.

Let's set up model parameters and run it. Initial *requirements* include half a million lines of code, *staff* number is 10 people, and each one's productivity includes 500 code lines per day. In this case, the project will be completed in 106 days (Figure 19.15). We can easily understand this result; since each developer produces 500 lines of code per day and all of them produce 5,000 per day, they would finish in 100 days, but we have 6% overhead on their communication.

We can get the same results by static modeling in Excel. The more details we add, the more difficult it is to get the same result by spreadsheet-based modeling. To expand the model further by adding a staff-hiring process that strongly affects total software development project time, let's continue thinking in terms of stocks and flows.

A major enhancement in system dynamics that was added in the first decade of the 21st century is hierarchical modeling. It allows you to create reusable components and use them on the main system dynamics diagram. This makes the system dynamics model easier to read, understand, and manage. Let's add a new object representing our development team into the model and use it as a subcomponent of the model.

The structure of our sub-model that represents a development team is shown in Figure 19.16. We hope you got used to system dynamics notation and AnyLogic.

Looking at the model in detail, there are two stocks "*new project staff*" and "*experienced staff*", and two flows representing new developers added into the project and their assimilation. By assimilation, we mean the process of training people to understand project details, scope, and boundaries.

Staff *assimilation rate* depends on the number of new people, which we should specify in the corresponding variable value. After 1 month of work, *new project staff* become experienced.

$$assimilation\_rate = \frac{new\_project\_staff}{month}$$

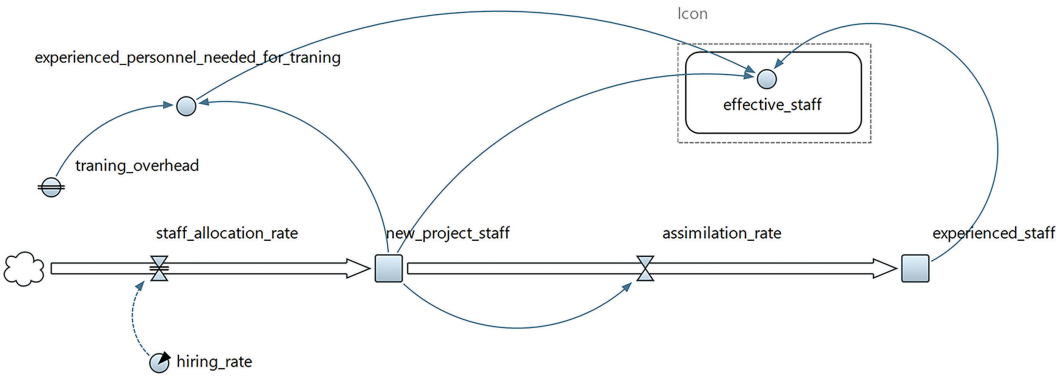

FIGURE 19.16 New staff allocation and assimilation—a subcomponent stock and flow diagram.

Experienced staff is used to train new employees, so there is *training overhead*. Let's assume that in our model, we need one experienced person working full time to train four newcomers.

Our sub-diagram has one interface variable called *effective staff*. The interface variable is visible on the upper level when you use the component. In our case, it represents a value of full-time experienced staff that can work on the project. We use the assumption that new people have 80% productivity and that only experienced people who are not involved in training activities can develop.

$$effective\_staff = 0.8 * new\_project\_staff$$
$$+ \left(experienced\_staff - experienced\_personnel\_needed\_for\_training\right)$$

Let's combine two models by adding the object "development team" onto the main diagram and connecting *effective staff* interface variable to *staff* variable. After the connection, the *staff* variable is driven by the *effective staff* variable (Figure 19.17).

Let's assume that in the same team of ten people, three new developers can be hired per month; to support hiring, we define *staff allocation rate* value.

$$staff\_allocation\_rate = 3.0/month$$

If we run the model, we see that the project is now finished in 83 days (Figure 19.18).

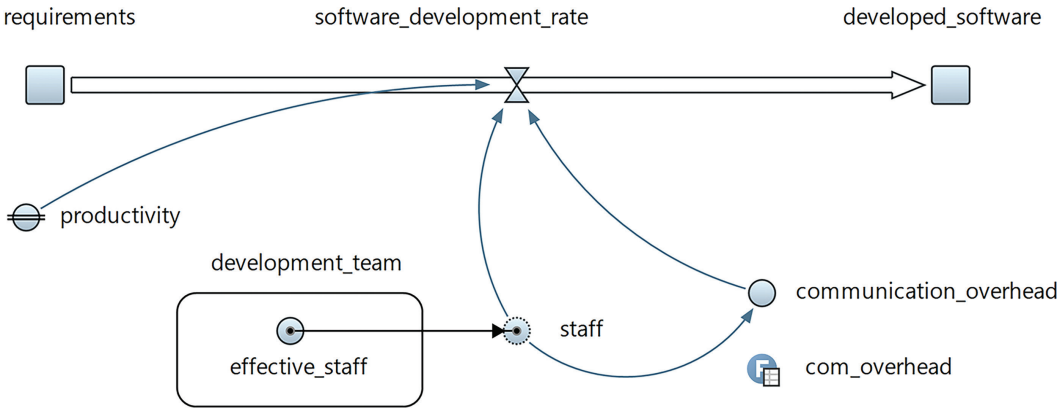

FIGURE 19.17 *Development team* sub-model embedded on to the main stock and flow diagram.

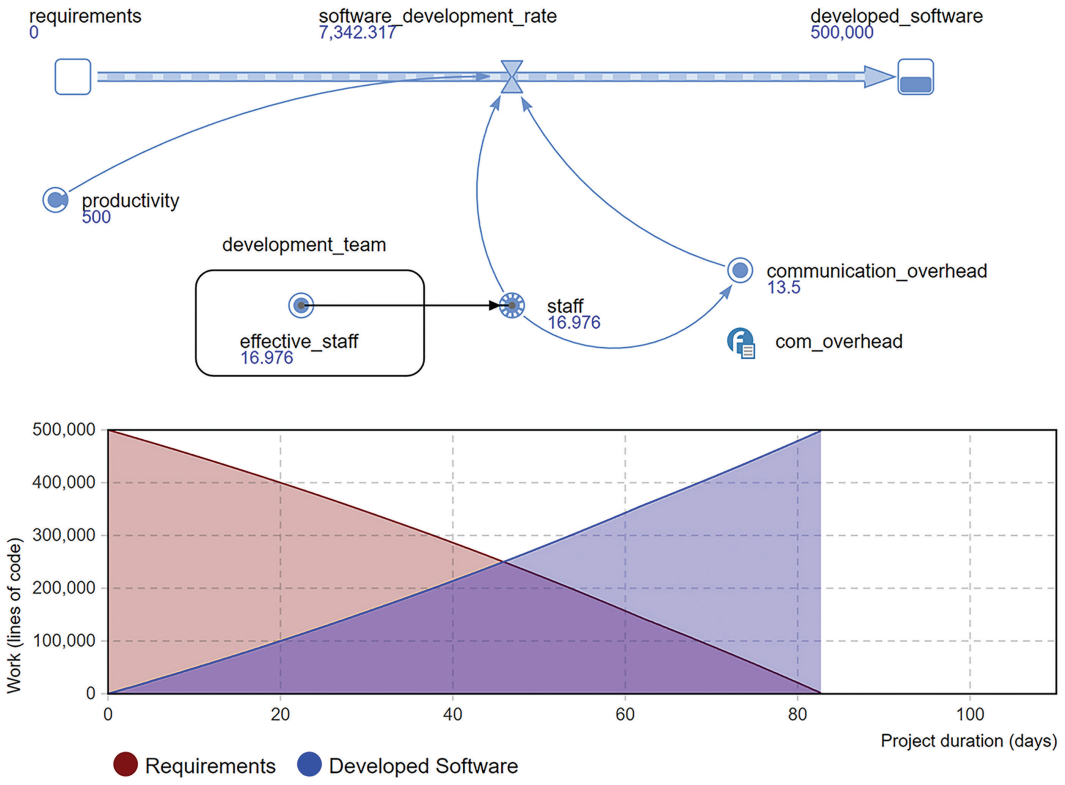

FIGURE 19.18 Modeling results, ten people, three new developers per month, 500,000 lines of code.

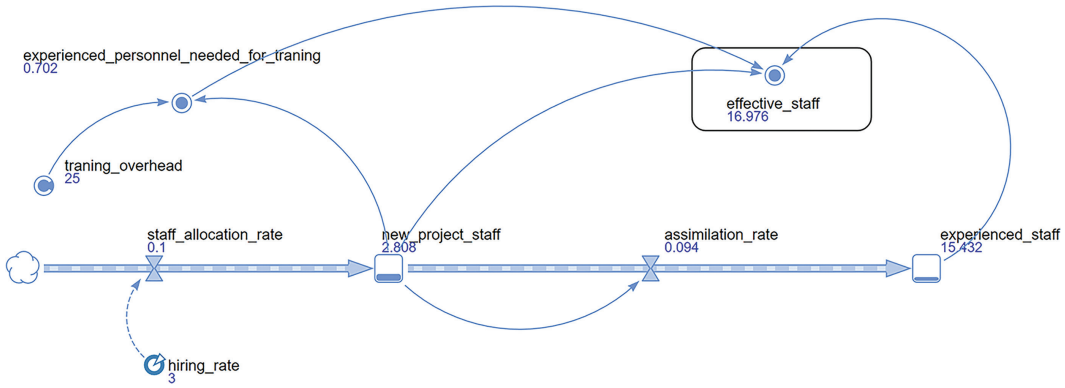

FIGURE 19.19 Modeling results, *development team* sub-model.

At the end of the project, we have a team of 15.432 experienced developers and 2.808 new employees, and their effective number is 16.976 (Figure 19.19). This is the typical situation for system dynamics, when we have fractional variable numbers representing people or other indivisible items. So we need to decide what having 2.808 new people on the team means. The reasonable assumption for this model would be to round this number up to 3, but for some other models, we just have to drop the fractional part of the figure.

Now the project manager has to understand everything about how a decision to add more people into the project influences project terms and costs. For simplicity, let's run another experiment to vary *hiring rate* from zero to twenty and create a plot of completion time versus hiring rate. As you can

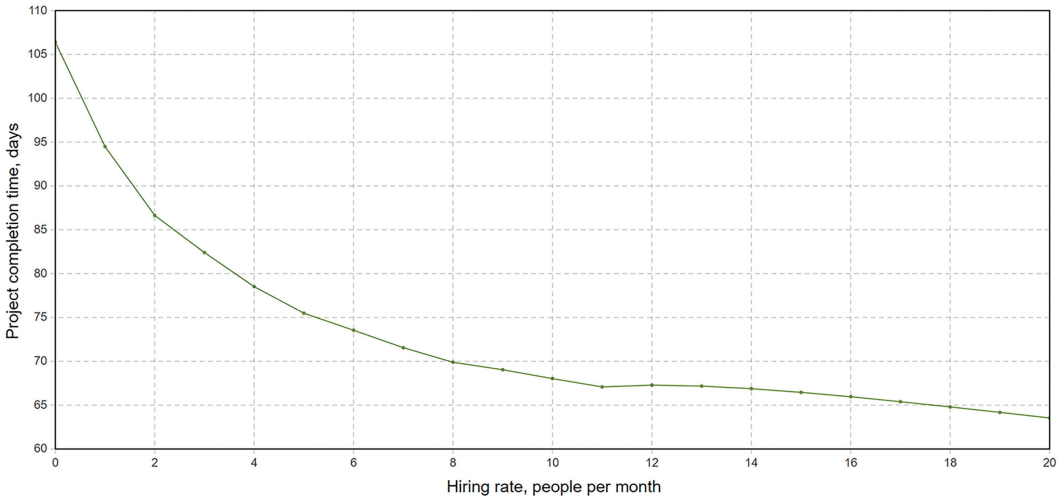

FIGURE 19.20 Hiring rate variation.

see in the plot in Figure 19.20, project completion time shows non-linear dynamics when we add new people to the project.

At this stage, we have not touched stochastics, which is the key benefit of simulation in project risk analysis. Let's do an experiment with our model setting *staff allocation rate* as a probability triangular distribution (Figure 19.21).

If we run the model 1,000 times and draw project completion time on a histogram, we will be able to analyze the risk of deadline failure. Assuming our deadline is 90 days, we know from simulation that if we allocate three new developers per month to our team, we finish the project in 83 days, a week ahead of schedule. When we add stochastic allocation rate, we see on the histogram that there is about a 28% probability that we do not meet the 90-day deadline; 27% is the sum of two last bars in Figure 19.22. The project manager should probably consider allocating more resources to the project, and we can continue experimentation with the model to support his decision-making.

To create this model, we used a general-purpose simulation software called AnyLogic. The model is publicly available in AnyLogic Cloud, along with the experiments we described and the model source code (it might be found by the name "Software development process dynamics" or by the author Sergey Suslov).[6]

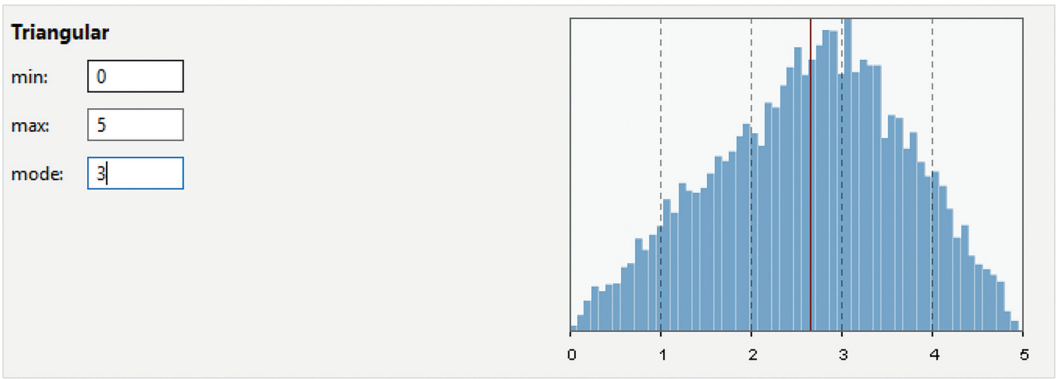

FIGURE 19.21 Triangular probability distribution.

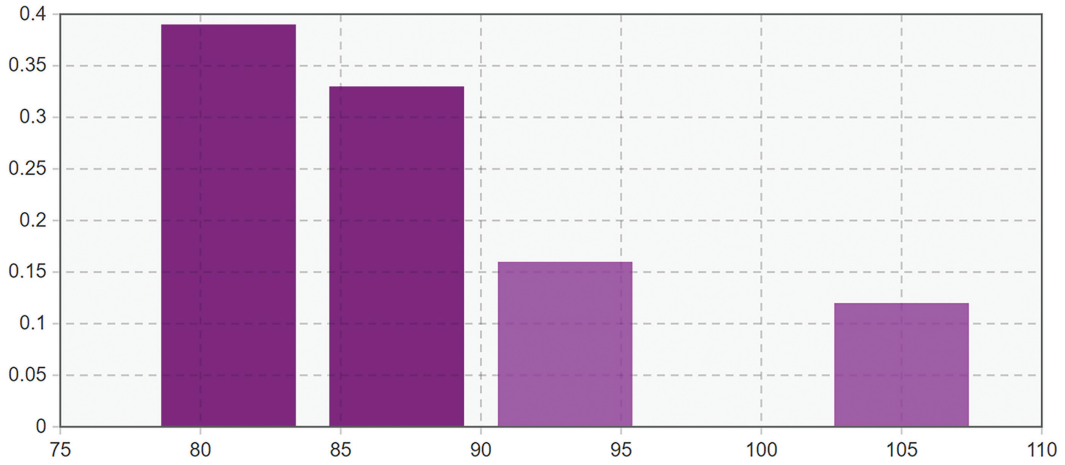

FIGURE 19.22 Histogram of project completion times in the case of stochastic *hiring rate*.

Agent-Based Modeling

Agent-based modeling (ABM) is the most recent major simulation method. Nevertheless, some models can be found in old papers and books like Thomas Shelling's 1978 book *Micromotives and Macrobehavior* (Schelling 1978). Such papers may be treated as the beginning of the ABM concept; for decades, ABM was limited in practical application by the state of computer hardware. In 1990–2000, ABM stayed a purely academic topic, but the 21st century with its booming computer hardware made ABM commercially applicable for solving high-scale business tasks; moreover, it now shows the fastest application growth compared with other simulation methods.

ABM and system dynamics are on different poles. While system dynamics is based on holistic process description, the ABM system uses a bottom-up approach that is described as interacting objects with their own behaviors. System behavior emerges as a summary of individual agents' actions. Scientific literature concentrates on the problems, such as what we can call an agent, whether or not it should have individual behaviors, memory, goals, etc. For the purpose of this chapter, we will concentrate only on the applied aspects of ABM.

In the past two decades, the ABM community has developed several practical ABM toolkits. In authors' opinion, the most popular tools are AnyLogic, Swarm, RePast, and NetLogo. Each toolkit has a variety of characteristics, its strong and weak points.[7]

To understand ABM in detail, let's try to reconceptualize our system dynamics software development model using ABM. We will continue using AnyLogic simulation software to develop new models.

The main object of an agent-based model looks very simple, as it includes only a couple of agent populations. A population is a set of agents, sometimes also called an array or collection. In our case, they are *developers* and *trainees*. The *Allocation* event periodically adds new *Trainees* to the system (Figure 19.23).

Since there is no place in an agent-based model where a global behavior is defined, let's start from the bottom with a *Developer* behavior. A statechart is a powerful tool to describe agent states and behaviors. Our *developer* and *trainee* agents' behavior can be graphically represented by the following statecharts (Figure 19.24).

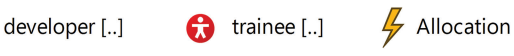

FIGURE 19.23 Agents collections and the event.

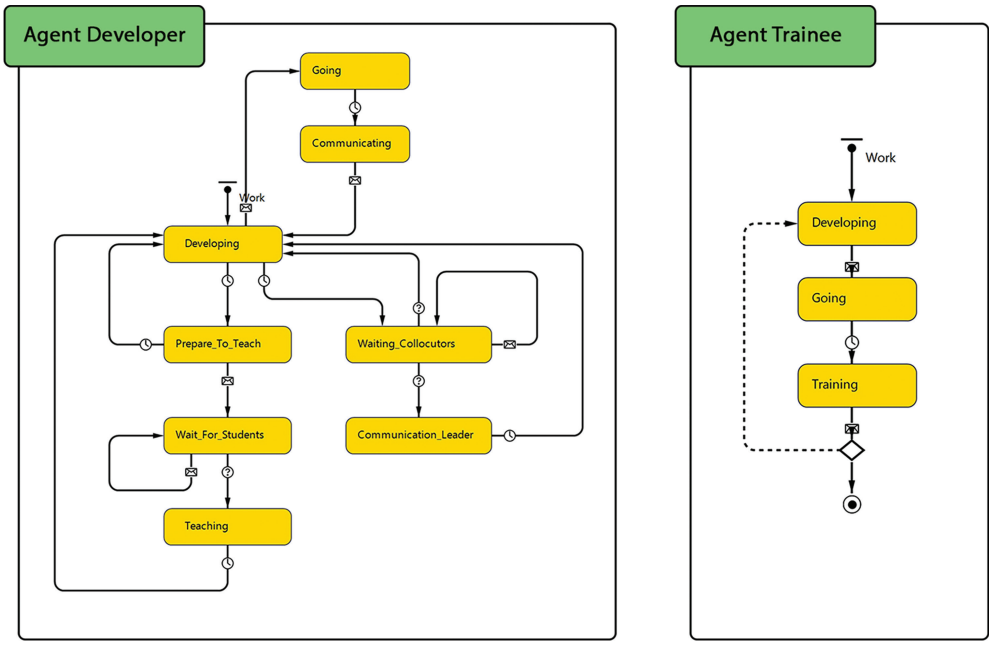

FIGURE 19.24 Statecharts that define agents' behavior.

Developing is the main state that the agent is in when starting its life in the model. Being in *Developing* state, the agent is producing the software product. *Teaching* is a state where a *Developer* agent does not develop but spends his time teaching new employees (represented by *Trainee* agent). There are two intermediate states between *Development* and *Teaching*, and they represent preparation for teaching and waiting for students. Agents communicate by sending messages, for example, in *Wait for students* state, a developer is periodically sending signals to all the trainees that it is in the room and ready to start the training.

Trainee logic is much simpler, there are only three states: *Developing* and *Training* which are clear, and *Going* which represents a break between development and training (it may be different in time depending on where the training room is located). One and the same model shows different results, depending on the spatial environment (whether all the developers and trainees are in one room, floor, building, city, country, etc.). To support a particular case, we should set up a timeout and determine how long the agent is spending in *Going* state. After each training, we check the experience of the trainee and decide whether to convert him to a developer. In this case, *Trainee* agent is disposed of and a new *Developer* agent is created.

Communication is also supported in the *Developer* agent's statechart. On a timeout, a developer decides to initiate a discussion and goes to *Waiting Collocutors* state, where it sends messages to its colleagues. When everybody is here, the agent becomes a *Communication leader*, leaving this stage after a given random timeout. Any *Developer* agent in *Developing* state switches to *Going* state on a corresponding signal from another agent who initiates the communication. When the agent becomes communication leader, it switches to *Communicating* state.

Agent-based models are usually full of code scripts, and in our case, Java language is provided by AnyLogic simulation software. A lot of small actions such as sending messages and agents' movements are done by adding a few lines of code into statecharts' transitions or states. For example, in our model we add new trainees by periodically executing three code lines.

If we execute this model, we get approximately the same results we would have gotten from the system dynamics model. You might wonder why we need ABM if we already have system dynamics. It is a very reasonable question, to which we have a set of answers:

- **Building agent-based models is more of a descriptive process.** We just describe how system components behave and communicate to get results. In a lot of cases, it is easier to describe the system components than to identify stocks, flows, and especially feedback loops.
- **ABM has a wider application range.** System dynamics is usually used for high-level management and organizational models, while ABM provides much better support for mid-level challenges such as supply chain simulation—where attention is paid to vehicles, distances, particular product stock levels on warehouses, etc.
- **ABM is a good tool to utilize company data.** Nowadays, companies have tons of data, and an agent-based model can use this data to get very precise model outputs. For instance, mobile network providers have their clients' full information (age, income, gender, etc.). They can model each client as an agent and get a precise digital twin of their client base or even a market segment in a given region.

AN AGENT-BASED CASE STUDY OF A SOFTWARE DEVELOPMENT COMPANY

There are many good case studies of ABM application for complex project management. For example, one big software development company used an agent-based model build with the approach described above to test its internal human resource management policies. They analyzed three alternative strategies to manage a growing global team:

- Internal education and coaching
- Replacing low-rated employees with new ones
- Combination of the strategies above.

This company does outsource projects, so the human resource part of the project is significantly important. They should always plan whether and how to grow the team, and develop its skills to have project managers, team leaders, and senior developer in place, along with regular staff that can be hired much easily. The agent-based model supports them in this planning.

Originally, most ABM applications assumed that the agent is a person: employee, customer, patient, etc. With growths in technology, the situation is rapidly changing; we see much more agent-based models where agents are objects, vehicles, warehouses, and even non-physical assets like projects. An agent-based case study of a logistics provider (see below) can help us to expand this idea, since the model includes such agents as vehicles, sites, etc.

AN AGENT-BASED CASE STUDY OF A LOGISTICS PROVIDER

Increasing competition in the logistics services market drives a demand for tools and solutions that make logistics operations more effective and efficient. A big 3PL logistics operator created an agent-based simulation model of over-sized cargo delivery over thousands of kilometers, to support a construction project in oil and gas industry.

The construction project was already planned, and a proposal was organized between logistics operators. The logistics operator sees calculating the precise cost of delivery and planning the logistics project as a big task. This is because there is a lot of freedom and small decisions, like what vehicles to use, what routes to take, and how to combine rail, road, and river transportation.

An agent-based model created in AnyLogic included such agent types as vehicle, base (intermediate point where the cargo can be stored), and construction plants (destination of the cargo). The model made allowance for what-if scenarios and also predicted how many vehicles were needed, where, and at what points in time.

The simulation model created with AnyLogic software supported logistics experts in

- Choosing the optimal form of multimodal transportation and considering lead times, transportation costs, and risks.
- Mitigating the financial risks and understanding the probability of satisfying expected lead time criteria.
- Forecasting the future state of the vehicles in short- and midterm perspectives.
- Budgeting and providing end customer with better contract conditions than other 3PL-operators
- Financial calculation in a short period for operational decision-making.

We hope that the case studies above helped show the diversity of agent-based applications for the management of complex projects. ABM can be applied at a high level of abstraction where most top management challenges occur, as well as at the middle abstraction level, where project details play a bigger role in emergent behavior.

Discrete Event Simulation

Discrete event simulation (DES) is almost as old as system dynamics. In October 1961, IBM engineer Geoffrey Gordon introduced the first version of GPSS (General Purpose Simulation System, originally Gordon's Programmable Simulation System), which is considered to be the first method of software implementation of discrete event modeling (Borshchev 2013). These days, discrete event modeling is supported by many software tools. From a practitioner's point of view, we should comment that when people apply simulation, the model is a discrete event or includes a discrete event subpart in over 50% of cases.

The main idea of DES is to consider the system as a process, that is, a sequence of operations being performed across entities. The operations include delays, service by various resources, choosing the process branch, splitting, combining, and so on. As long as entities compete for resources and can be delayed, queues are present in virtually any discrete event model. The model is specified graphically as a process flowchart, where blocks represent operations. The flowchart usually begins with "source" blocks that generate entities and inject them into the process and ends with "sink" blocks that remove entities from the model. This type of diagram is familiar to the business world as a process flowchart and is ubiquitous in describing their process steps.

All major DES tools support the same set of basic flowchart blocks such as source, queue, delay/process/service, and sink. There are more than 50 different DES tools,[8] some of them are general purpose such as Arena, AnyLogic, Simul8, and ExtendSim; some are industry-specific, for example, Siemens Plant Simulation, FlexSim, and Automod. Any general-purpose software tool can be easily applied for solving project management challenges using DES.

Let's think about how we can create a model of our software project management using DES. First of all, we need a source for our requirements and sink for developed software. As soon as we have requirements, we need to seize a developer from *experienced staff*. Then, we process a piece of the requirements and release a developer (Figure 19.25).

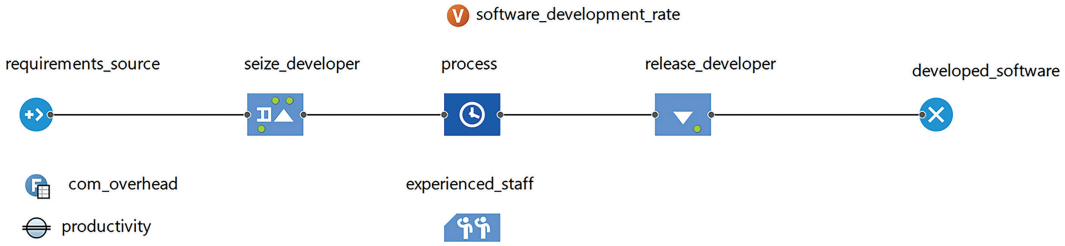

FIGURE 19.25 Software development model implemented as a DE flowchart.

Let's feed the model with data. Requirements source should produce 500,000 lines of code, each of them an entity. Let's introduce a new variable called *software development rate* and assign it the following value:

$$software\_development\_rate = \left(\frac{1 - com\_overhead\left(experienced\_staff.size()\right)}{100} \right) * productivity$$

By doing this, we decrease the software development rate on communication overhead, exactly as we did in system dynamics. We can reuse the table function *com_overhead*.

We should set up the *process* time as 1/*software_development_rate*, because this is a time in days that one developer needs for one line of code. We have a team of ten developers so that our *process* can process up to ten lines of code simultaneously.

Running the model, we get the same 106 days for project completion time as we got by applying system dynamics (Figure 19.26).

Let's think about how to implement new developer allocation. We can start by adding a new, initially empty resource pool called *new project staff*. We should specify that we can use this pool when we *seize developer* (Figure 19.27).

A scripting language is a necessary evil of the professional model-building process that some simulation practitioners like the most and some others struggle with the most. Scripting makes modeling tools very powerful and flexible, but not so easy to create and manage. In our model, we use Java code expression to set up unique *software_development_rate* for each line of code processed in process block. This expression is executed each time we seize a developer.

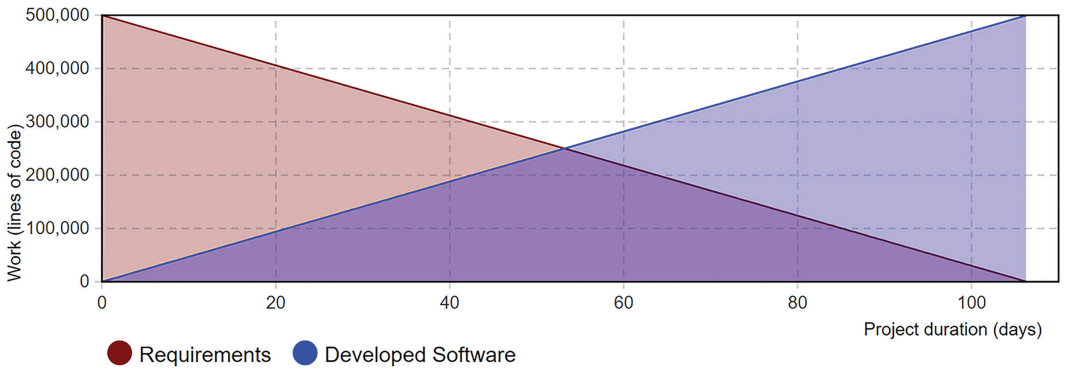

FIGURE 19.26 DE model results, 500,000 lines of code, ten people.

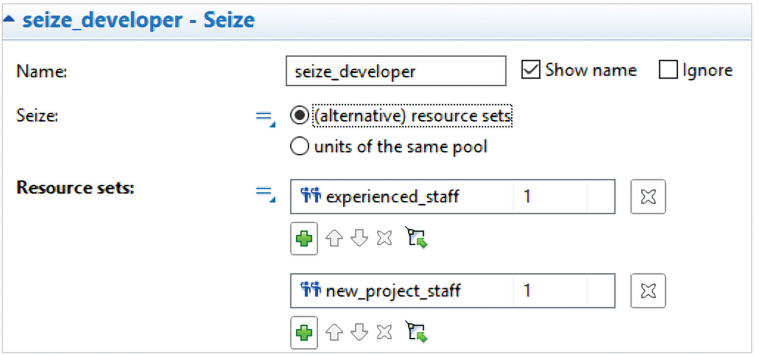

FIGURE 19.27 Seize block settings in AnyLogic software tool.

Then, let's modify the Java code that we execute on seizing the developer:

```
software_development_rate = productivity*
(1-com_overhead(experienced_staff.size()+new_project_staff.size())/100)*
(((Developer)unit).experienced? 1 : 0.8)*
(1-training_overhead/100*new_project_staff.size()/experienced_staff.size());
```

Software development rate is assigned a new value that equals productivity decreased by three multipliers:

- The first one to support communication overhead
- The second one to change productivity in case the seized developer is a new employee
- The last multiplier to support that we have to allocate experienced people time for training.

By this expression, we added *new project staff* into communication overhead, supported training overhead calculations, and changed productivity for new developers. Processing speed will be unique or each particular entity representing a line of code, depending on the developer and current team size and structure.

The last and most important part is the allocation process for new developers (Figure 19.28). As the most important part of the processes, it starts with *Source* block and ends with *Sink* block. New developers leave *staff allocation* source block with *hiring rate* per month. As soon as a new developer appears, we increase the capacity of *new project staff* resource pool. After that, new developers enter *training* service block, where they immediately start monthly training. After the *training* ends, we increase *experienced_staff*, adding new resources and decreasing *new_project_staff*.

When we run the model with the same parameters, we get slightly different results caused by the discrete nature of *staff allocation*. Project completion time is 88 days, and at the end of the project, we have 15 experienced developers and 3 new people (Figure 19.29).

FIGURE 19.28 DE model results, 500,000 lines of code, ten people.

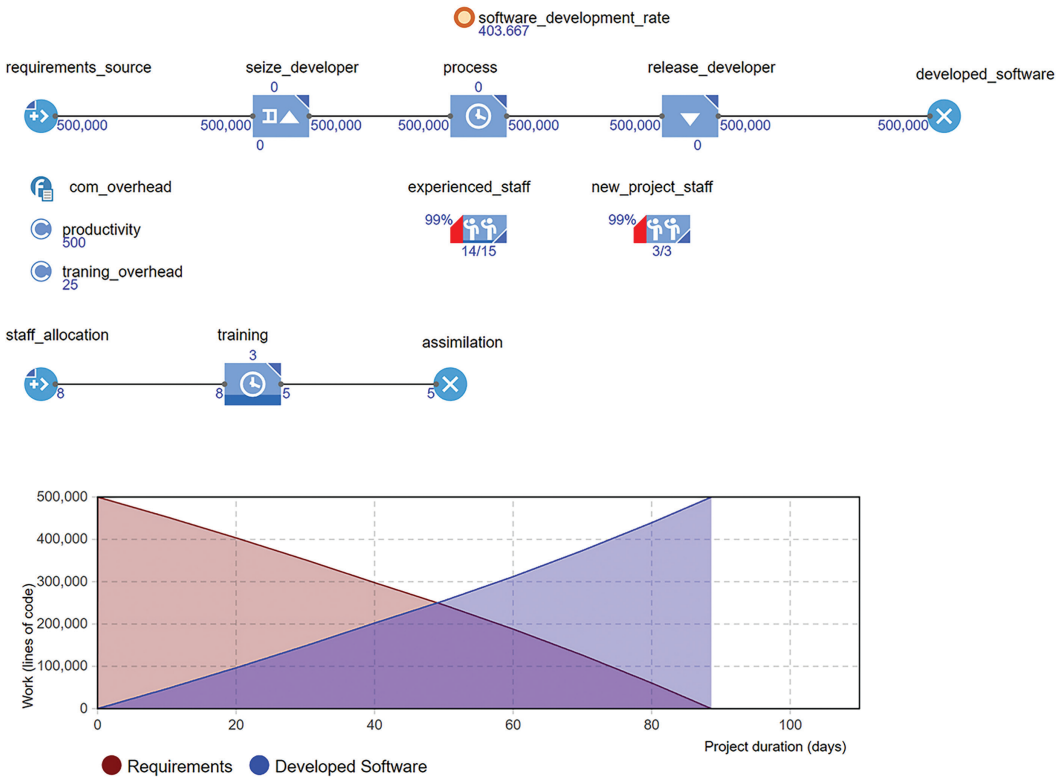

FIGURE 19.29 DE model results, 500,000 lines of code, ten people, three new developers per month.

The model along with its source code is published in AnyLogic Cloud,[9] and you can find it by searching by the name "Software Process Dynamics DE" or by the author name Sergey Suslov.

You can use free AnyLogic PLE[10] to see the full model internals or even continue the development.

DESs are usually used for the engineering part of project management, where managers and engineers try to predict the completion time of every project stage and mitigate the risk of being out of schedule due to technical reasons. In comparison with system dynamics or ABM, DES is used more on low to middle abstraction levels, where detailed business or technology processes play key roles. The case study below should help to visualize how operational constraints influence the feasibility of the project goals.

A CONSOLIDATED CONTRACTORS COMPANY CASE STUDY

Let's look at the case study of Consolidated Contractors Company (CCC), which is the largest construction company in the Middle East and ranks #18 internationally. CCC has offices and projects in over 40 countries and a workforce of over 130,000 employees. Its portfolio includes oil and gas plants, refineries and petrochemical facilities, pipelines, power and desalination plants, light industries, water and sewage treatment plants, airports and seaports, heavy civil works, dams, reservoirs and distribution systems, road networks, and skyscrapers. Just after signing a contract for a site preparation project (earthmoving scope of approximately four

million cubic meters), the client and local authorities placed new, more restrictive constraints on the operation. These constraints included

- Trucks were allowed to move at a maximum of 10 kmph within the construction project site instead of the original 20 kmph.
- Trucks from/to the dump location were instructed to follow a specific route full of traffic lights, intersections, roundabouts, and security gates. Accordingly, the original assumption of the average 40 kmph truck speed on route from and to dump site could not be maintained.
- Truck sizes/loads were brought down from the maximum allowable of 32–15 m^3.
- The number of truck trips restricted to a maximum of 100/h.
- Only one of the original four site access points was granted after signing the contract, thus restricting all traffic on site to one gate.
- Only one work shift (10 h/day) was allowed for dumping at the dump site in contrast to the original two-shift (20 h/day) schedule.

The newly placed constraints essentially meant that the project schedule would be severely impacted, as would the total cost of completing it. The number and complexity of the constraints made it quite difficult to manually evaluate the impact of the constraints on time to complete equipment requirements. As a result of the newly added complexity, the simulation team was asked to help quantify the impact of the new constraints and substantiate a claim by CCC for a time extension. CCC already had a well-established construction simulation model made in AnyLogic software. This model was used for basic construction optimization, to forecast equipment and time requirements for earthworks operations. Using the earthworks simulator requires multiple parameter inputs, including the assumed average speed the trucks will travel on their haul routes and back. With a new truck route imposed on the project, it was very difficult to manually estimate the average speed the trucks would run at.

As such, a more complex construction simulation model was built in AnyLogic software to mimic the trucks traversing the route segments, while both loaded and empty. Using optimization techniques in AnyLogic was helpful for risk-free experimentation and for effective construction management on this project. In the construction simulation model, each route segment was modeled with a stochastic distribution for the total time to traverse the segment. Then, the trucks were made to run 10,000 times each way in the simulator to arrive at an average speed for each route and loaded/empty combination.

Why use simulation in construction? DES with AnyLogic was specifically selected for construction modeling because this software allowed CCC to

- Very quickly build a construction optimization model representing the route and its segments.
- Add a map of the routes and superimpose an animation of the trucks traversing the routes to make it easier to visually explain the work to stakeholders.

The average speeds deduced from the truck route simulator were then fed into the earthworks construction simulation model along with the remainder of the new constraints. These included

- Truckload sizes and the number of work shifts (working hours) per day, to produce multiple scenarios showing the original time and equipment requirement forecast to complete the operations.

- The current time and equipment requirement forecast to complete the operations (impact quantification in time and resource requirements).
- Proposed mitigation scenario.

The results of the two-step process of using the truck route simulator to summarize the route and then feeding it into the earthworks simulator enabled CCC to quantify very quickly the impact of the new more restrictive constraints and to build mitigation scenarios to aid the claim for extension of time. Both of the construction simulation models were created in AnyLogic software, giving them the flexibility to consider all the new constraints. DES has proven to be good assistants for construction management. Based on the construction simulation model results, the client agreed to extend the total duration of the earthworks operation by 50% on top of the original schedule duration and to allow two work shifts per day. This essentially saved the project an estimated additional cost equivalent to 18% of the original total contract value.

Combined Approach: Key Challenges and Advantages

Even though we were able to create one and the same model of software project management using three different simulation methods, it is rare in practice. Often you can choose a modeling method following practical recommendations that can be found in the Internet:

- If there are many independent objects, use an agent-based approach.
- If there is only information about global dependencies, use system dynamics.
- If a system is easily described as a process, use a discrete event approach (Figure 19.30).[11]

Most real-world cases are complex, and it is convenient to describe different parts of a system with different methods. In using only one method, your ability to capture business systems with their real complexity and interactions may be seriously limited. Some system elements will have

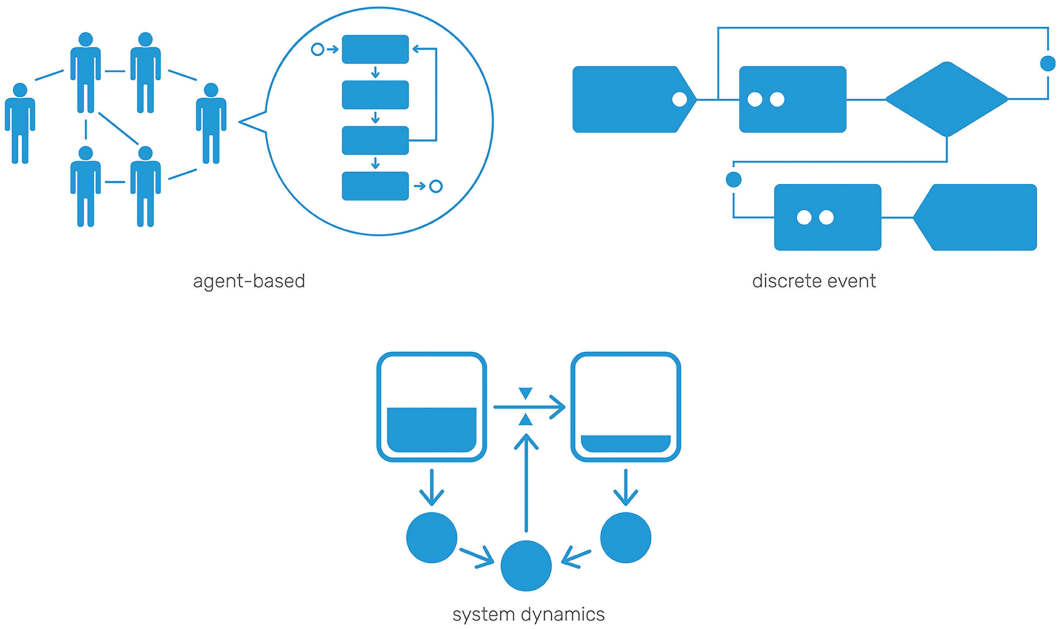

FIGURE 19.30 Simulation methods.

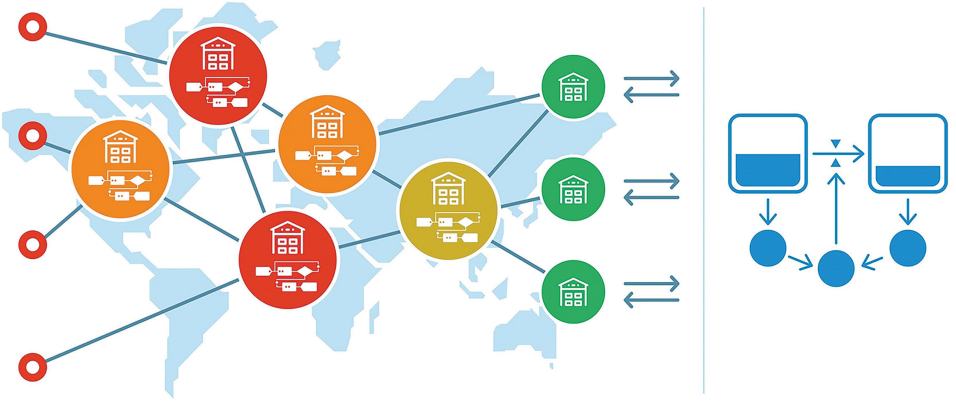

FIGURE 19.31 Supply chain multimethod model.

to be excluded or a workaround developed. In such cases, it makes sense to consider mixing different simulation approaches in one model, which is called multimethod or hybrid simulation.

Most of the actively developing simulation vendors have started implementation of the support for at least some multimethod modeling architectures. It is likely that moving forward only multimethod simulation will survive as an industry standard for general-purpose simulation software tools. In *The Big Book of Simulation Modeling* (Borshchev 2013), Andrei Borshchev suggested a set of typical multimethod architectures including

- **Agents interacting with a process model.** For instance, equipment defined as an agent influences the process of a conveyor system.
- **A process model linked to a system dynamics model.** A business process flowchart can change the variables of a system dynamics diagram for a company's profit and loss statement.
- **System dynamics inside agents.** Production processes inside factories can be defined with system dynamics (SD) and used as agents in a supply chain model.

Here, in Figure 19.31 you can see how production, distribution, and the market can be combined in one model, using different techniques. A discrete event model describes the processes within each warehouse. The warehouses then appear as agents on the distribution network. Finally, the market, which drives the system, is modeled with system dynamics. Everything is captured without compromise.

- **Processes inside agents.** A good example are the warehouses in Figure 19.31.
- **Agents temporarily act as entities in a process.** For instance, a regional healthcare model, where people defined as agents go to hospital and follow a discrete event flowchart.

We recommend you follow technical recommendations given in this book during project management challenges conceptualization phase.

QUESTIONS

1. Why is it beneficial to use simulation for complex projects? What advantages does simulation provide?
2. Why do projects fail from the simulation experience perspective?

3. What is counter-intuitive in the system behavior of complex projects?
4. What are ineffective remedies that managers typically use in practice? Why do these measures fail to fix the project?
5. How would you suggest fixing project management problems applying a systemic perspective?
6. What are the key sources of risk in the management of complex projects?
7. What are the main modeling approaches for the simulation of complex projects? What are their differences, advantages, and drawbacks?
8. What practical recommendations would you provide to a project manager?
9. What in your opinion prevents project managers from widespread use of simulations? Why? Please, explain your point of view.

CONCLUSION. SIMULATIONS: WHY BOTHER?

Simulation modeling offers a holistic view of the project. The overall complexity of even not-so-large projects quickly exceeds human capacity to rationally assess dynamics of the system. Herbert Simon, an economics Nobel Prize laureate, introduced a concept of "bounded rationality" to emphasize this phenomenon (Simon 1972, 1979; Dörner 1996). Simulations help to produce counter-intuitive conclusions. For instance, Lee et al. (2007) investigated and found out that contrary to the widespread assumption that everything should be done as quickly as possible to minimize delays, projects may have optimal delay sizes that are inevitable but still do not significantly hurt the overall project performance. Experienced project managers intuitively account for the possibility of small project delays.

Simulations are the right tool to make such an assessment holistically and simulate various scenarios. Unlike traditional approaches, simulations can incorporate many subjective factors such as staff morale, productivity, motivation, fatigue, deterioration of quality, and many others that are usually not explicitly analyzed. Over the years, system dynamics has developed a language and methodology for expressing various tangible and non-tangible variables (factors) that can be quantified (numerical estimates can be provided) and added to the model.

A number of studies document a successful use of simulations to support various projects. For example, naval shipbuilding (Cooper 1980), highway construction (Ford et al. 2004), semiconductor cheap development process (Ford and Sterman 1998), flour production (Godlewski et al. 2012), military projects (Lyneis et al. 2001), software project management (Abdel-Hamid and Madnick 1982), and construction projects (Pena-Mora and Park 2001; Love et al. 2002).

The advances in simulation software now help building models in the presence of management easier and faster. MIT professor John Sterman explains:

> Traditionally, formal modeling tools were complex and inaccessible to all but the trained analyst. As a result, models were developed by experts without direct involvement in the modeling process …. The inability of the managers [who were expected to use the results of the model] to participate in the process created a dilemma. If the modelers built a simple model, they were criticized for ignoring important relationships. If they built a complex model, they were criticized for creating a black box no one could understand. …. The solution to the dilemma is the intensive involvement of the management team in the modeling process. New software tools now make it possible for managers to participate as full partners in model development. Sophisticated interfaces and intuitive designs allow managers and employees throughout a firm to use, test, and revise complex models.
>
> **Sterman (1992: 11–12)**

A group model building (GMB) methodology was developed by Richardson and Andersen (1995, 2010), Vennix (1996), Lane (2000), Andersen et al. (2007), and others. Originally, such workshops were conducted as 1- to 2-day seminars where experienced simulation practitioners together with top-management representing the client company used to discuss the conceptual structure of the system and build a model afterwards. Lately, the practice evolved into what is currently known as

"participatory system modeling" (Van den Belt 2004, Stave 2010, and others) "… which encompasses GMB and mediated modeling, but places more emphasis on seeing participation along a continuum of model building and formulation, from no involvement to high involvement" (Hovmand 2014: 17).

Group model-building workshops can be very different in terms of the type of model to be developed—from a simple map providing a first glance into the project to a comprehensive computer simulation model. Even simple maps typically used at the early stages of a modeling process prove to be effective in conceptualizing a problem and providing useful insights into the key feedback structure of the modeled project. GMB workshops proved to be very useful in terms of higher quality of the model, since a model is built with the help of facilitation teams—who are usually experienced simulation practitioners—and deep expertise of industry practitioners—who are not necessarily experts in simulation modeling. In addition, GMB with early involvement of clients or key stakeholders helps to get a buy-in from stakeholders and propagate practical model application by end-users at a later stage.

When prepared, a simulation can be used at various stages of the project in a number of ways. For instance, Lyneis et al. (2001) suggest using simulations for project management practitioners:

- **At a pre-project stage** for bid or plan analysis, risk assessment, and competitor analysis (to estimate using information from public sources, if possible, what the program can cost competitors)
- **During the project stage** for risk management, change management (analysis of potential impact in terms of costs and scheduling specification and work scope changes from customer), and evaluation of process changes (total impact of changes to evaluate trade-off between short-term costs and long-term benefits realized)
- **After the project is finished** for benchmarking, training, and developing personnel (as a management flight simulator to communicate project experience and lessons learnt to the management).

RECOMMENDATIONS FOR PROJECT MANAGEMENT PRACTITIONERS

- It might be rather useful to view project management, not as a sequence of discrete tasks but *flows of work*, much of which consists of a Rework cycle. It's important to have a *rigorous* and *timely error-discovery process* to reveal errors soon enough that their correction is done quickly and with minimal effort. Early resolution significantly cuts disruption.
- Managers can influence project performance; however, they should take into account the *complex* and *counter-intuitive nature of major feedback loops* involved. Avoid conventional wisdom (i.e., "Let's hire more") and apply system thinking.
- Beware of *self-reinforcing feedback loops* (overtime, rapid hiring, conflicting deadline pressures, etc.) which quickly spiral a project out of control. It is important to understand the sources of the vicious circles, to maintain control over project costs.
- Usually, it is much cheaper to *move schedule pressure* (if possible) than to try to meet deadlines.
- More and later changes do not just create more impact; they create them disproportionately.
- Indirect project disruption costs are far more dangerous, typically resulting in costs up to ten times higher than direct cost overruns.
- *Try to budget for 10%–50% overspending* to be able to mitigate early problems quickly—major project disruption can easily lead to extra costs *times* the original project budget.
- *Try to model and simulate:* the theory and practice are already here.

Simulation is a powerful tool for project managers. With the advancements in simulation tools and dissemination of knowledge in theory and practice, it becomes easier and cheaper to simulate complex projects.

As project practitioners get more experience in the modeling and simulation of complex projects, we will experience less project failures and cost overruns, making the project management industry less prone to errors.

Simulation is a powerful tool for project managers. With the advancements in simulation tools and dissemination of knowledge in theory and practice, it becomes easier and cheaper to simulate complex projects thus considerably saving money and efforts of project practitioners.

ENDNOTES

1 Productivity of a software development project can be measured in various ways, that is, lines of source code produced per certain amount of time, number of bugs per 1,000 of code lines, active days (the time spent by a programmer to develop a code, not including time for planning and other minor activities), tasks scope (a volume of code a programmer can deliver yearly), and others. Some also use cycle time (order is taken for production and complete) and lead time (order is received from customer and delivered to him) as productivity metrics. For convenience purpose, in this publication, we will measure progress in *lines of source code produced per certain amount of time* (although industry practitioners generally do not believe this is an effective metric to measure software development productivity).

2 http://vensim.com/ (accessed March 12, 2019).

3 The term suggested by Jay Forrester, the founder of the system dynamics method.

4 "The rework cycle tends to delay project staffing, pushing the actual staff profile to the right, as errors are created, discovered, and reworked. This creates more errors which are discovered and reworked. Vicious circle feedbacks triggered by management responses to project problems tend to increase staffing, pushing project staffing (and effort applied through overtime) up, and to the right. This is due to productivity losses and increasing errors as a consequence of delays in recognizing and responding to project problems and the rework cycle" (Ford et al. 2007: 9).

5 www.anylogic.com (accessed March 12, 2019).

6 The direct link to the model https://cloud.anylogic.com/model/14482b77-2be0-4a80-bfb9-2f8c20ea627f?mode=SETTINGS (accessed March 12, 2019).

7 Comparison of agent-based modeling software: https://en.wikipedia.org/wiki/Comparison_of_agent-based_modeling_software (accessed March 12, 2019).

8 Simulation Software Survey, ORMS Today: www.informs.org/ORMS-Today/OR-MS-Today-Software-Surveys/Simulation-Software-Survey (accessed March 12, 2019).

9 The direct link to the model: https://cloud.anylogic.com/model/3fbdf014-76f2-4393-b194-cad66dd12381?mode=SETTINGS (accessed March 12, 2019).

10 www.anylogic.com/downloads/personal-learning-edition-download/ (accessed March 12, 2019).

11 www.anylogic.com/use-of-simulation/multimethod-modeling/ (accessed March 12, 2019).

FURTHER READING

Cooper, K. G. 1980. Naval ship production: A claim settled and a framework built. *Interfaces* 10, no. 6 (December): 20–36. doi:10.1287/inte.10.6.20.

Cooper, K. G. 1994. The $2,000 hour: How managers influence project performance through the rework cycle. *Project Management Journal* 25, no. 1: 11–24.

Godlewski, E., G. Lee and K. Cooper. 2012. System dynamics transforms fluor project and change management. *INFORMS Journal on Applied Analytics* 42, no. 1: 17–32. doi:10.1287/inte.1110.0595.

Howick, S. 2003. Using system dynamics to analyse disruption and delay in complex projects for litigation: Can the modelling purposes be met? *The Journal of the Operational Research Society* 54, no. 3: 222–229. doi:10.1057/palgrave.jors.2601502.

Lyneis, J., O. de Weck and S. Eppinger. 2003. Course notes for MIT course ESD.36J: System and project management (Fall 2003). https://dspace.mit.edu/handle/1721.1/80702 (accessed March 12, 2019).

Lyneis, J. M., K. G. Cooper and S. A. Els. 2001. Strategic management of complex projects: A case study using system dynamics. *System Dynamics Review* 17, no. 3 (Autumn): 237–260. doi:10.1002/sdr.213.

Lyneis, J. M. and D. N. Ford. 2007. System dynamics applied to project management: A survey, assessment, and directions for future research. *System Dynamics Review* 23, no. 2–3 (Summer–Autumn): 157–189. doi:10.1002/sdr.377.

Reichelt, K. and J. Lyneis. 1999. The dynamics of project performance: Benchmarking the drivers of cost and schedule overrun. *European Management Journal* 17, no. 2 (April): 135–150. doi:10.1016/S0263-2373(98)00073-5.

Repenning, N. and J. Sterman. 2001. Nobody ever gets credit for fixing problems that never happened: Creating and sustaining process improvement. *California Management Review* 43, no. 4 (Summer): 64–88. doi:10.2307/41166101.

Rodrigues, A. and J. Bowers. 1996. System dynamics in project management: A comparative analysis with traditional methods. *System Dynamics Review* 12, no. 2 (Summer): 121–139. doi:10.1002/(SICI)1099-1727(199622)12:2<121::AID-SDR99>3.0.CO;2-X.

REFERENCES

Abdel-Hamid, T. K. 1993. A multiproject perspective of single-project dynamics. *Journal of Systems Software* 22, no. 3: 151–165. doi:10.1016/0164-1212(93)90107-9.

Abdel-Hamid, T. and E. Madnick. 1982. *Modeling the Dynamics of Software Project Management*. Working paper, Alfred P. Sloan School of Management (February 1988, #WP 1980–88). https://dspace.mit.edu/bitstream/handle/1721.1/48430/modelingdynamics00abde.pdf?sequence=1 (accessed March 12, 2019).

Andersen, D. F., J. A. M. Vennix, G. P. Richardson and E. A. J. A. Rouwette. 2007. Group model building: Problem structuring, policy simulation and decision support. *The Journal of the Operational Research Society* 58, no. 5 (May): 691–694.

Boehm, B. W. 1981. *Software Engineering Economics*. Upper Sadle River, NJ: Prentice-Hall PTR.

Borshchev, A. 2013. *The Big Book of Simulation Modeling: Multimethod Modeling with AnyLogic 6*. La Vergne, TN: Lightning Source Inc.

Chan, M. 2011. Fatigue: The most critical accident risk in oil and gas construction. *Construction Management and Economics* 29, no. 4: 341–353. doi:10.1080/01446193.2010.545993.

Cooper, K. G. 1980. Naval ship production: A claim settled and a framework built. *Interfaces* 10, no. 6 (December): 20–36. doi:10.1287/inte.10.6.20.

Cooper, K. G. 1993a. The rework cycle: Why projects are mismanaged. *PM Network* 7, no. 2: 5–7.

Cooper, K. G. 1993b. The rework cycle: How it really works and reworks. *PM Network* 7, no. 2: 25–28.

Cooper, K. G. 1993c. The rework cycle: Benchmarks for the project manager. *Project Management Journal* 24, no. 1: 17–21.

Cooper, K. G. 1994. The $2,000 hour: How managers influence project performance through the rework cycle. *Project Management Journal* 25, no. 1: 11–24.

Cooper, K. G., J. M. Lyneis and B. J. Bryant. 2002. Learning to learn, from past to future. *International Journal of Project Management* 20, no. 3 (April): 213–219.

Cooper, K. G. and K. S. Reichelt. 2004. Project changes: Sources, impacts, mitigation, pricing, litigation, and excellence. In *The Wiley Guide to Managing Projects*, ed. P. W. G. Morris and J. K. Pinto, 743–772. Hoboken, NJ: Wiley. doi:10.1002/9780470172391.ch31.

Dörner, D. 1996. *The Logic of Failure: Recognizing and Avoiding Error in Complex Situations*, trans. R. and R. Kimber. New York: Basic Books.

Eden, C., T. Williams and F. Ackermann. 2005. Analysing project cost overruns: Comparing the "measured mile" analysis and system dynamics modelling. *International Journal of Project Management* 23, no. 2 (February): 135–139. doi:10.1016/j.ijproman.2004.07.006.

Eick S. G., C. R. Loader, M. D. Long, L. G. Votta and S. V. Wiel. 1992. Estimating software fault content before coding. In *ICSE '92 Proceedings of the 14th international conference on Software engineering (Melbourne, Australia—May 11–15, 1992)*, 59–65. New York: ACM. doi:10.1145/143062.143090.

Ford, D. N., S. Anderson, A. Damron, R. de Las Casas, N. Gokmen, and S. Kuennen. 2004. Managing constructability reviews to reduce highway project durations. *Journal of Construction Engineering and Management* 130, no. 1 (February): 33–42. doi:10.1061/(ASCE)0733-9364(2004)130:1(33).

Ford, D. N., J. M. Lyneis and T. R. B. Taylor. 2007. Project controls to minimize cost and schedule overruns: A model, research agenda, and initial results. In *Proceedings of the 25th International Conference of the System Dynamics Society and 50th Anniversary Celebration*, ed. J. Sterman, R. Oliva, R. S. Langer, J. I. Rowe and J. M. Yanni, 1–27. Albany, NY: System Dynamics Society. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.488.4883&rep=rep1&type=pdf (accessed March 12, 2019).

Ford, D. N. and J. Sterman. 1998. Dynamic modeling of product development processes. *System Dynamics Review* 14, no. 1 (Spring): 31–68. doi:10.1002/(SICI)1099-1727(199821)14:1<31::AID-SDR141>3.0.CO;2-5.

Godlewski, E., G. Lee and K. Cooper. 2012. System dynamics transforms fluor project and change management. *INFORMS Journal on Applied Analytics* 42, no. 1: 17–32. doi:10.1287/inte.1110.0595.

Graham, A. K. 2000. Beyond PM 101: Lessons for managing large development programs. *Project Management Journal* 31, no. 4: 7–18. doi:10.1177/875697280003100403.

Holzmann, G. 2009. Appendix D – Software complexity. In *NASA Office of Chief Engineer. NASA Study of Flight Software Complexity: Final Report*, www.nasa.gov/pdf/418878main_FSWC_Final_Report.pdf (accessed March 12, 2019).

Holzmann, G. J. 2007. Conquering complexity. *Computer* 40, no. 12 (December): 111–113. doi:10.1109/MC.2007.419.

Hovmand, P. S. 2014. *Community Based System Dynamics*. New York: Springer Science + Business Media. doi:10.1007/978-1-4614-8763-0_2.

Lane, D. C. 2000. Diagramming conventions in system dynamics. *The Journal of the Operational Research Society* 51, no. 2 (February): 241–245.

Larson, W. J. and J. R. Wertz. 1993. *Space Mission Analysis and Design*. 2nd ed. Torrance, CA: Microcosm.

Lee Z. W., D. N. Ford and N. Joglekar. 2007. Resource allocation policy design for reduced project duration: A systems modeling approach. *Systems Research and Behavioral Science* 24, no. 6 (November/December): 551–566. doi:10.1002/sres.809.

Love, P. E. D., G. D. Holt, L. Y. Shen, H. Li, and Z. Irani. 2002. Using systems dynamics to better understand change and rework in construction project management systems. *International Journal of Project Management* 20, no. 6 (August): 425–436. doi:10.1016/S0263-7863(01)00039-4.

Lyneis, J. M., K. G. Cooper and S. A. Els. 2001. Strategic management of complex projects: A case study using system dynamics. *System Dynamics Review* 17, no. 3 (Autumn): 237–260. doi:10.1002/sdr.213.

Lyneis, J. M. and D. N. Ford. 2007. System dynamics applied to project management: A survey, assessment, and directions for future research. *System Dynamics Review* 23, no. 2–3 (Summer–Autumn): 157–189. doi:10.1002/sdr.377.

NASA Office of Chief Engineer. 2009. NASA study of flight software complexity: Final report. www.nasa.gov/pdf/418878main_FSWC_Final_Report.pdf (accessed March 12, 2019).

Neves, F. G., H. Borgman and H. Heier. 2016. Success lies in the eye of the beholder: The mismatch between perceived and real IT project management performance. In *49th Hawaii International Conference on System Sciences*, 5878–5887. Los Alamitos, CA: IEEE Computer Society.

Pena-Mora, F. and M. Park. 2001. Dynamic planning for fast-tracking building construction projects. *Journal of Construction Engineering and Management* 127, no. 6 (December): 445–456. doi:10.1061/(ASCE)0733-9364(2001)127:6(445).

Richardson, G. P. and D. F. Andersen. 1995. Teamwork in group model building. *System Dynamics Review* 11, no. 2 (Summer): 113–137. doi:10.1002/sdr.4260110203.

Richardson, G. P. and D. F. Andersen. 2010. Systems thinking, mapping, and modeling in group decision and negotiation. In *Handbook of Group Decision and Negotiation*, ed. D. M. Kilgour and C. Eden, 313–324. Dordrecht: Springer Verlag.

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

Roberts, E. B. 1964. *The Dynamics of Research and Development*. New York: Harper & Row.

Roberts, E. B. 1974. A simple model of R&D project dynamics. *R&D Management* 5, no. 1: 1–15. doi:10.1111/j.1467-9310.1974.tb01217.x.

Rodrigues, A. and J. Bowers. 1996. The role of system dynamics in project management. *International Journal of Project Management* 14, no. 4 (August): 213–220. doi:10.1016/0263-7863(95)00075-5.

Schelling, T. C. 1978. *Micromotives and Macrobehavior.* New York: W. W. Norton & Co.

Simon, H. A. 1972. Theories of bounded rationality. In *Decision and Organization: A Volume in Honor of Jacob Marschak*, ed. C. B. McGuire and R. Radner, 161–176. Amsterdam: North-Holland.

Simon, H. A. 1979. Rational decision making in business organizations. *The American Economic Review* 69, no. 4 (September): 493–513.

Stave, K. 2010. Participatory system dynamics modeling for sustainable environmental management observations from four cases. *Sustainability* 2, no. 9: 2762–2784. doi:10.3390/su2092762.

Stecklein, J. M., J. Dabney, B. Dick, B. Haskins, R. Lovell and G. Moroney. 2004. Error cost escalation through the project life cycle. Conference paper. NASA Johnson Space Center. https://ntrs.nasa.gov/search.jsp?R=20100036670 (accessed March 12, 2019).

Sterman, J. 1992. System dynamics modeling for project management. http://scripts.mit.edu/~jsterman/docs/Sterman-1992-SystemDynamicsModeling.pdf (accessed March 12, 2019).

Taylor, T. and D. N. Ford. 2006. Tipping point failure and robustness in single development projects. *System Dynamics Review* 22, no. 1 (Spring): 51–71. doi:10.1002/sdr.330.

The AnyLogic Company. 2016. Developing disruptive business strategies with simulation. www.anylogic.com/resources/white-papers/developing-disruptive-business-strategies-with-simulation/ (accessed March 12, 2019).

Van den Belt, M. 2004. *Mediated Modeling: A System Dynamics Approach to Environmental Consensus Building.* Washington, DC: Island Press.

Vennix, J. A. M. 1996. *Group Model Building: Facilitating Team Learning Using System Dynamics.* Chichester: Wiley.

Williams, T., C. Eden, F. Ackermann and A. Tait. 1995. The effects of design changes and delays on project costs. *Journal of the Operational Research Society* 46, no. 7 (July): 809–818.

Williams, T. M. 1999. The need for new paradigms for complex projects. *International Journal of Project Management* 17, no. 5 (October): 269–273. doi:10.1016/S0263-7863(98)00047-7.