7

CURRENT AND FUTURE DIRECTIONS

7.1 INTRODUCTION

There are many important problems and challenging opportunities for extensions of the work discussed in this book, as well as for the overall field of software process simulation. In general, the current and future directions presented here apply to all forms of simulation for software processes. System dynamics will be one of several techniques to further the theories and practice of software engineering as will discrete event methods, agent-based modeling, and others. The different techniques will also be used in combination with each other more frequently. System dynamics will be integrated with other process modeling aspects, similar to how software product architecture descriptions require multiple perspectives.

This chapter overviews some current (and presently apparent) future directions as of early 2006, but the list is certainly incomplete as new issues and technologies will arise. One thing we can be sure of is continued rapid change in the field. It is also inevitable that simulation will become more pervasive and integrated with everyday processes. This will be driven by advances in modeling and the practical need to be able to respond to changes.

Figure 7.1 shows a stack of software process simulation technologies, from the supporting infrastructures through simulation development environments and up to advanced new modes of simulation operations. Everything is built upon infrastructure software represented in the bottom tier, and the level of domain knowledge and sophistication of simulation increases up the stack. Like computer science in general, some

Software Process Dynamics. By Raymond J. Madachy

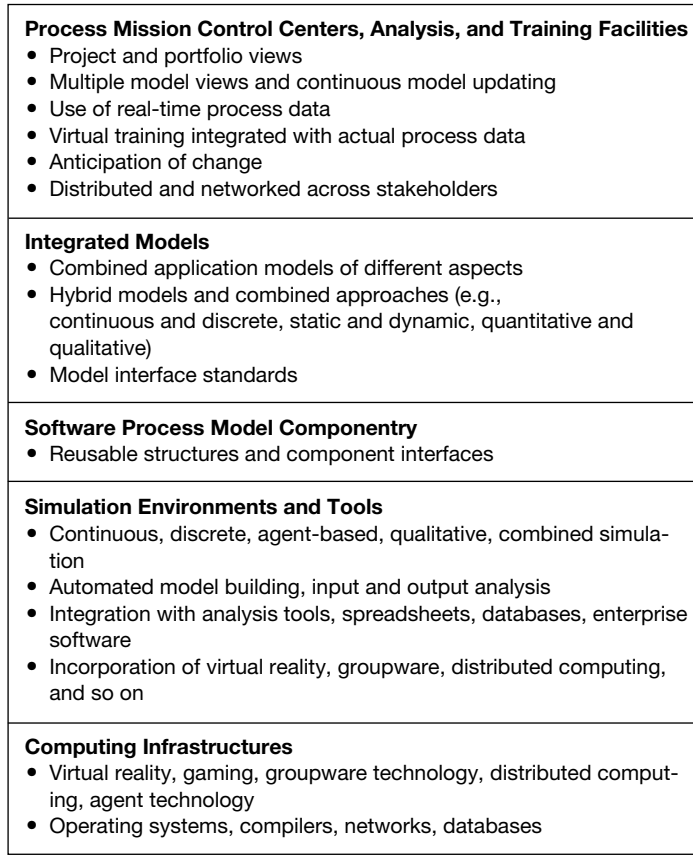

Figure 7.1. Software process simulation technology stack.

people create new operating systems and compilers upon which new applications are built, whereas others specialize in particular application areas. There are back-and-forth dynamics between the layers as new infrastructure creates new application opportunities, and application needs may fuel new infrastructure technology.

The technology layer for simulation environments and adjunct tools requires the basic computing infrastructures underneath. Applications are then created with the simulation environments, and may optionally be built using reusable model structures and components for the software process domain. The application models may be further combined with other applications or even other types of modeling techniques. The top layer represents advanced simulation operations in terms of how models are used, combined with other project management methods, or how they leverage more of the infrastructure technologies to increase simulation utility.

This chapter will describe future directions in all but the bottom infrastructure layer, which is outside the scope of process simulation, but it will assume continued advancements in the infrastructure areas. The middle tiers generally follow the book pro-

gression from simulation fundamentals to the higher-level applications based on the fundamentals. Part 1 of this book focuses on the modeling process, tools to a small extent, and the middle layer of software process model componentry. Part 2 looks at some applications for continuous systems modeling represented in the middle tier and a few integrated models belonging in the next tier up. The applications can be further integrated with other models of various types and brought into more advanced usage represented by the top two layers.

Simulation environments and tools will continue to advance in capabilities for automated model building, visualization, output analysis and optimization, model integration, and distributed usage across the Internet or other (sub) networks. Artificial intelligence and knowledge-based techniques will be used for some of the automation functions. Future environments that automate many aspects of model building will help minimize the effort for model development.

The trends of increasing computing power, distributed calculations across networks, and virtual reality provide new and unique advantages for users. These will lead to simulation software that employs parallel and distributed simulation techniques to execute computations and provide graphics-intensive simulations. Computer games and other software applications keep improving virtual reality to provide more interactive and lifelike simulation. The degree to which users can immerse themselves into a virtual environment will continue to grow.

Software process model componentry involves developing reusable modeling components across the industry (both macro- and microprocess views), refinement and enhancement of specific model structures, and incorporating object-oriented techniques with modeling structures. Application models will become easier to construct as componentry advances are made. Improved simulation environments will leverage the reusable components and provide more application-specific tools for software process aspects.

Future application models will be driven by new and emerging trends in software processes. These overlapping trends include increased globalization and distributed development, more open-source development and improved COTS, increased emphasis on the user and people concerns, new agile and hybrid processes, increased product demands for aspects such as security and quality, more complex systems of systems, and personnel shortfalls in skills and talent to meet the new demands. Increasingly rapid change permeates all of the above trends. Additionally, more of the applications will come from a value-based perspective and models will become more interdisciplinary, encompassing broader enterprise concerns.

Models will become more integrated not only in terms of different application areas, but also with respect to modeling techniques. For example, the need for combined continuous and discrete models is already apparent and the practice of hybrid modeling is gaining momentum. Other meta-modeling techniques such as analytic equations or heuristic approaches will be integrated with and/or compared to simulations. The practice of using different meta-models will support theory building in the field of software engineering.

It is also incumbent on modeling practitioners to improve their processes in conjunction with the advanced environments and new directions. For example, better modeling methods and empirical research to demonstrate model validation would help achieve wider acceptance of process simulation in the software industry.

Simulation and empirical software engineering methods will become more strongly connected. There is much synergy between them as each can help advance the other. As empirical research becomes more integrated with simulation, they will be used together for theory development and validation in the software field.

Simulation will become more commonly practiced and inherent in the software process. Models will not be so detached from everyday operations but instead will become a natural way of doing things. For example, project dashboards will include models and simulations and evolve into process "mission control" centers that are model-intensive. They will be used constantly for "what if" analyses and to respond to unanticipated changes. The models will be updated continuously in light of changing assumptions and more accurate data from the actual process, and they may also change as a result of machine learning from them.

With technology advances and improved data collection, many fields now depend on extensive use of simulation as a way of doing business, to test new theories and options. Other disciplines are also evolving their simulations into advanced modes of operation with the improved environments and computing infrastructures. The software process discipline will naturally progress as the other fields have. The rest of the chapter will describe some exciting current and future directions for applications and research, and how many of them are inextricably related to each other.

7.2 SIMULATION ENVIRONMENTS AND TOOLS

Advancing technology will lead to exciting new features in simulation environments and tools. There will be a host of new and improved ways to build models, simulate, interact with the simulations in stimulating ways, optimize them in a goal-driven manner, and automatically extract useful information from the simulation runs. There are more hybrid simulation approaches and toolsets being used. Tools are becoming better at integrating representation, guidance, simulation and execution capabilities.

Other disciplines have at least partially brought their simulations into advanced modes of operation, most notably war game simulations. They are currently the most complex simulations of man-made systems; they combine different simulation techniques in hybrid modeling approaches, they model entities in the thousands or even millions, and frequently use distributed computing and advanced supercomputing capabilities to crunch the numbers. They also use virtual reality and have a long history of using simulation for game playing and training.

The U.S. Defense Modeling and Simulation Office (DMSO) uses advanced modeling and simulation research that incorporates many of the concepts in this chapter [DMSO 2006]. They foster interoperability, reuse and affordability of modeling and simulation. They have developed a general-purpose simulation High Level Architecture (HLA) to support those goals. It is adopted as the standard architecture for all Department of Defense simulations and also for distributed simulation systems by the Object Management Group (OMG). Some of the ideas and standards from this domain can potentially be brought into the software process modeling domain. See [DMSO 2006] for extensive information and resources on simulation including the HLA.

The next subsections describe some new and future aspects of simulation environments and tools. They include recent and frequently limited examples of their application to software process modeling and simulation.

7.2.1 Usability

Even with modern computer tools, system dynamics can sometimes require a hefty intellectual investment, particularly if one has a nontechnical background. The Ithink, Vensim, and Powersim tools offer highly usable human interfaces, but the task of modeling becomes more difficult when domain knowledge comes into the picture. That is a reason why automated model building for specific domains would be a quantum leap of capability. Knowledge-based techniques can help, per Section 7.2.3, as can model componentry, per Section 7.3, and related methods such as system dynamics metamodeling for software processes [Barros et al. 2006a] (Section 7.3.2).

Some of the provided models in this book can be enhanced for usability by adding interactive control-panel capabilities. To support policy analysis and experimentation, user-friendly slide controls for varying model parameters, buttons, and dials can be employed with the system dynamics simulation software packages. Such an interface is more along the lines of a virtual dashboard for project control, per Section 7.7, Process Mission Control Centers.

Simulation tools, like many other software applications, will continue to evolve to become collaborative platforms for people to work in groups. New user-interface issues will arise as groupware technology becomes integrated into simulation environments.

Simulation tools will become easier for nontechnical people to use, much like spreadsheets have. However, there is a potential downside that the quality of these simulations may decrease as amateurs "do their own thing" without proper assistance [Nance 2000]. This argues for robust, automated, and application-specific simulation toolsets to reduce the risk of poor modeling as well as changing educational curriculums to introduce simulation to a wider variety of students.

7.2.2 Model Analysis

Techniques to analyze models, such as sensitivity analyses, were demonstrated in previous chapters. The study by [Houston et al. 2001a], for example, took the general approach of sensitivity analysis to compare and understand four different system dynamics software process models. However, new simulation environments will come with increased statistical support and experimental design facilities to ease the burden. There are advanced methods for evaluating models based on optimization and machine learning techniques described in this section and the next.

The verification and validation of models will always be challenging, and to some extent be a partial roadblock to widespread acceptance of simulation. Recent work addressing this problem in [Wakeland et al. 2005] illustrates the use of heuristic algorithms to improve the verification and validation of software process simulation models using system dynamics. In this approach, an optimization problem is formulated to guide a heuristic search algorithm that attempts to locate particular combinations of

parameter values that yield surprising results. These results often help the modeler to identify flaws in the model logic that would otherwise remain undetected.

Advanced techniques for data mining or machine learning are other forms of model analysis. Machine learning is the ability of programs to improve performance over time by extracting rules and using them to solve new problems. Machine learning was not applied to software process simulation models until recent advancements in tools such as TAR2 [Menzies et al. 2002, Menzies et al. 2004a] made it possible. In these applications, treatments are learned from the models that improve the simulation outputs.

Treatment learners find the fewest factors that most influence a simulation model. The TAR2 tool is a data miner for performing automatic sensitivity analysis of large data files, looking for constraints or parameters that can most improve or degrade the performance of a system. Numerous Monte Carlo simulations are run, attribute values are ranked, treatments are built with a small number of attribute ranges with high rankings, and the new treatments are tested in fresh simulation runs.

Some examples of using machine learning on models for software cost estimation and risk assessment can be found in [Menzies, Richardson 2005]. TAR2 is applied to the static COCOMO cost estimation model [Boehm et al. 2000] and the Expert COCOMO heuristic risk analyzer [Madachy 1997]. At USC, we are currently applying machine learning with the ODC defect model highlighted in Chapter 5 to assess risks for NASA flight projects.

Genetic programming is another machine learning technique that uses an evolutionary optimization algorithm [Koza 1992]. It imitates genetic algorithms, which use mutation and replication to produce algorithms that represent the "survival of the fittest." The integrated use of genetic programming and system dynamics modeling for software processes is described in [Ramesh, Abdel-Hamid 2003]. An application is shown of a decision support system enhanced with a genetic algorithm for optimizing quality assurance allocation.

7.2.3 Artificial Intelligence and Knowledge-Based Simulation

Simulation and artificial intelligence (AI) both try to model reality to solve problems, and can contribute to each other. Improvements in expert systems can potentially affect all aspects of simulation. Knowledge-based simulation (KBS) is the application of AI to simulation operations. Expert systems can be developed for a large list of tasks related to simulation. These include the use of AI knowledge representation techniques for the modeling of complex systems as well as the codification of simulation expertise to manage the simulation life cycle [Fox et al. 1989]. Intelligent simulation environments include automation of different simulation tasks, including model construction, verification, output analysis, experimental design, and documentation. See [Fox et al. 1989] for an overview of an artificial intelligence approach to system modeling and automating the simulation life cycle using knowledge-based techniques.

An interdisciplinary approach to modeling and simulation that incorporates the representational flexibility and ease of use that AI techniques offer is provided in [Weidman et al. 1989]. Another framework for integrating simulation and AI based on their similarities is described in [Doukidis, Angelides 1994].

Expert systems already exist for automating various simulation tasks such as checking input consistency, elaboration of high-level models for specific domains, statistical analysis and experimental design, analysis of output to determine model relationships, suggesting changes, and others.

A variety of enhancements and further research can be undertaken for the knowledge-based aspects of software process models. One drawback of system dynamics modeling tools is that they are relatively insular, and not easily amenable to integration with other tools such as expert system shells. A challenge is to overcome this current limitation and integrate them into unified applications.

It is unlikely that knowledge-based aids will completely automate simulation. There will always be a need for humans to interpret the output and apply contextual judgment. Since problems keep changing, there will always be new situations that automated solutions cannot handle.

7.2.4 Networked Simulations

The Internet has already revolutionized working modes in many regards, and network technology may provide substantial benefits for advanced uses of simulation. Simulation across the Internet includes models running in a distributed mode at multiple servers/clients, models accessing data across the Internet, models running at a server being accessed by users, and combinations of these modes [Jain 1999].

Many of the major system dynamics tool vendors already have some mechanism for running their simulations over the Internet [isee 2006, Powersim 2006, Ventana 2006], and some exclusively offer Web-based simulations [Forio 2006].

Improvements in distributed simulation coincide with project trends for distributed development in multiple locations. Not only can each location be simulated independently on a network, but each location can have its own respective simulations. For example, the hybrid model proposed in [Raffo, Setamanit 2005] is distributed in the sense that each development location has separate discrete-event submodels and separate system dynamics submodels. There is also a global continuous submodel for project-level dynamics in the proposed approach.

7.2.5 Training and Game Playing

Virtual simulation for training and game playing will become more common in the future, though it is already ubiquitous in aerospace and military applications. Most of the models presented in this book have been run in a detached mode without extensive user interaction. The mode of executing models by themselves (like batch jobs) is called constructive simulation. However, virtual simulation with people in the simulation loop can be quite powerful. The combination of system dynamics models with an immersive simulation technology is intriguing. Virtual reality and agent-based technologies have the potential to better integrate humans in the simulation loop.

Some experiences with simulation for personnel training were summarized in Chapter 4. Strong results from empirical experiments in [Pfahl et al. 2004b] demonstrate that a simulation-based role-playing scenario is a very useful approach for learn-

ing about issues in software project management. Favorable training results were also reported in [Collofello 2000] and [Madachy, Tarbet 2000].

An overview of issues in learning and decision support for software process management is described in [Pfahl et al. 2006b]. Examples in the automotive industry illustrate how simulation can become a useful management tool for the exploration and selection of alternatives during project planning, project performance, and process improvement. Also discussed are limitations, risks, and proposed future work.

The development of games for improving and enriching a student's learning experience is again on the rise. The beer game [Sterman 1989] in the field of system dynamics was developed to instill the key principles of production and distribution. Some recent game applications for software process applications are [Barros et al. 2006a], SimSE [Oh Navarro, van der Hoek 2005, Birkhoelzer et al. 2005], and SimVBSE [Jain, Boehm 2005].

A system-dynamics-based game for software project management was developed in [Barros et al. 2006a]. It describes a game intended for training purposes and the changes that were made to allow a system dynamics simulator to support game-like interaction. An experimental evaluation of the game's application to management students was performed, and they proposed models to describe the story underlying a game without programming.

SimSE is an educational software engineering simulation game that uses both predictive and prescriptive aspects [Oh Navarro, van der Hoek 2005]. It provides a simulated game with virtual office scenes for its players to take on the role of a project manager and experience the fundamentals of software engineering through cause–effect models. It supports the creation of dynamic, interactive, graphical models for software engineering process education. The authors have built an initial simulator of the waterfall lifecycle process to teach some process principles to students.

SimVBSE is a rule-based game for students to better understand value-based software engineering and its underlying theory [Jain, Boehm 2005]. It is also a medium to test hypotheses about value-based engineering theories. In SimVBSE, the player traverses through different rooms, gets briefed by virtual employees, and gets to choose various options, including changing project parameters, making strategic investments, and applying processes. The player gets to see relevant project metrics and his decisions involve the consideration of stakeholder value. Cause and effect rules are invoked and the student gets an assessment for his actions in each scenario.

Even though these simulation games are oriented toward educational applications, they can also be adapted for actual project environments (see Section 7.7, Mission Control Centers). Extensive use of simulation for training is also undertaken in military applications. See [DMSO 2006] for resources and information on use of simulation for training in a defense context.

7.3 MODEL STRUCTURES AND COMPONENT-BASED MODEL DEVELOPMENT

Chapter 3 introduced common structures that are reusable and adaptable for a variety of situations. The essential model elements are fixed, but all the generic flow process-

es, infrastructures and chains can be further investigated and refined. Chapters 4–6 presented models with some suggestions for modifications, and some of the changes are described in the chapter exercises. Many of the proposed future directions for the models involve revising model structures or creating new ones. Virtually all of the models could be enhanced to "drill down" further into various subprocesses. The reader is encouraged to look back at those chapters for more ideas on specific model componentry.

A framework for component-based model development would provide a higher level of abstraction for modelers, and could accelerate model creation. Development of models would be made easier, substantial effort could be saved, and modeling errors reduced. The reusable modeling structures presented in Chapter 3 are not yet provided in a "drag-and-drop" tool interface for easy incorporation into models. Making such structures more accessible for component-based modeling is a future challenge. Work in this area may lead to new language possibilities and simulation paradigms.

Methods for developing models with easily composable parts would be beneficial. Some model structures could become standard "plug and play" modules across the simulation community. A desired model could be developed with off-the-shelf elements or generation aids, similar to applications composition practices that rapidly compose programs from interoperable components, or applications generation whereby domain-specific programs are generated from high-level specifications. For example, a robust defect chain could be instantiated for a particular life-cycle process working at a high level of abstraction. The tedium of producing a defect model from the ground up and thoroughly testing it would be largely eliminated.

Component-based development takes advantage of patterns. All software programs are composed of patterns [Gamma et al. 1995]. So is process knowledge itself [Chang 2005], and simulation models are no exception. Entire models are thus constructed from basic patterns and the rules for putting them together can also be expressed as patterns. Software process patterns would be encapsulated as reusable building block components (see Section 7.3.2 on metamodels as an example).

The components would be assembled together with the help of predefined interfaces that describe their respective inputs and outputs, and modelers would simply "wire" them up accordingly. The predefined components serve as starting points and are not always cookie-cutter solutions, because new situations would frequently require their adaptation.

There are a variety of ways to ease the integration of model pieces. Making components easy to interface requires careful architecting. Interface specifications would have to be developed that enable disparate structures to be hooked together in a dynamic model. Some applied examples of submodel integration are shown in [Johnson 1995] using model variables common to multiple sectors of the Abdel-Hamid model as integration linchpins.

The GQM technique could be used to rigorously define data structures for models to adhere to. Simple examples of applying GQM to system-dynamics-based software process models were shown in Chapter 2, and [Pfahl et al. 2002] discusses a framework for using GQM to help integrate modeling with descriptive process and goal-oriented measurement. Also see the work on metamodels in Section 7.3.2 for composing and integrating software process models at a high level.

Submodels do not have to be directly connected if other data exchange methods are used. For example, XML is a possible way to integrate stand-alone submodels using it as an interchange standard to exchange data. XML would serve as a flexible glue between submodels while allowing substitution of other submodels that adhere to the XML schemas. XML is also discussed in Section 7.5.1 for unifying larger independent models. See [Diker, Allen 2005] for a proposed XML standard for system dynamics models.

Issues of reuse in software process modeling were described in [Neu, Rus 2003]. The authors looked at possible approaches in software engineering that may transfer over to simulation modeling. Some assets they identified that may be reusable include requirements, environment, scenarios, static process models or their components, influence diagrams, relations between process parameters, model design (patterns), executable model elements, and modeling knowledge. They also described ways to facilitate reuse in development environments.

7.3.1 Object-Oriented Methods

Object-oriented methods are a natural and convenient way to support component-based model development, and system dynamics lends itself to object orientation, as briefly discussed in Chapter 3. System dynamics structures and their behavior can be easily likened to classical "objects" with associated data items and methods. Elaborated models are comprised of instantiations of objects with levels, rates, and methods between them consisting of flow integration logic. (Note that in the section below on metamodels, "classes" are used in a more traditional object-oriented and fine-grained way to represent sets of elements such as "developers").

In object-oriented simulation, modular objects are used that encapsulate attributes and procedures. The objects are dynamic data types with fields and methods (procedures) that pass messages to invoke methods or update attributes. There are also inheritance of attributes/methods and polymorphism properties associated with them. These principles were shown in the Chapter 3 class hierarchy and demonstrated with example model components. Advantages of object-oriented simulation include easier reuse of process modules, modification of models, general ease of use, and reduction of modeling errors [Khoshnevis 1992].

7.3.2 Metamodels

Advanced work in the area of metamodels has been performed by Barros and colleagues [Barros et al. 2001b, Barros et al. 2002a, Barros et al. 2006b]. Metamodels are extensions to system dynamics that allow the development and specialization of domain models, providing a high-level representation for developers within domains. A domain expert develops a model, which conveys the relevant categories of elements that compose the domain and the relationships among those elements. A developer uses this model to describe a particular problem, by specifying how many elements of each category exist in the model of interest and the particular characteristics of each

one. Finally, the model is translated to system dynamics constructors in order to be simulated and analyzed.

The authors have developed a metamodel that allows the development of software process models based on high-level constructors instead of mathematical equations. These constructors represent software process domain concepts such as developers, activities, resources, and artifacts. A domain model allows the translation of these concepts to traditional stock-and-flow diagrams, which can be simulated to evaluate the behavior of software process models. The approach helps inexperienced modelers build process models by reducing the semantic gap between system dynamics and the software process domain.

7.4 NEW AND EMERGING TRENDS FOR APPLICATIONS

Writing this book has amply demonstrated how dynamic the field is, as it was difficult to keep up with new and emerging trends in software technology, processes, and the business environment. This section identifies anticipated trends likely to impact the types of applications covered in Chapters 4–6. What is currently written as new and emerging trends may soon become the focus of robust, working models or be overtaken by other unprecedented trends.

Underlying all of this is the continued trend of more rapid change. The ability to adapt to change becomes ever more important for individuals and organizations alike. When added to the trend toward emergent requirements, the pace of change places a high priority on process agility and investments in continuous learning for both people and organizations. A major challenge in organizations is to determine which legacy processes and principles to keep, modify, or eliminate. The pace of change will inflict heavy penalties on overly bureaucratic and document-intensive software processes [Boehm 2005a].

The accelerated change is driven by technology trends such as Moore's Law about increasing computer capacity, the continuing need for product differentiation, and quick-to-market business strategies. Global connectivity also accelerates the ripple effects of technology, marketplace, and technology changes. Rapid change increases the priority of speed over cost to meet market windows. Simulation can be used to evaluate these trade-offs for new products (see the value-based product model in Chapter 6) as well as identify and prepare for major changes to legacy systems.

Ubiquitous change also prompts significant changes in software engineering education. Students should not only learn concepts, processes, and techniques, but also to learn how to learn [Boehm 2005]. Simulation and game technology are some powerful strategies for helping students learn how to learn, as described in section 7.2.5, Training and Game Playing. Fortunately, the educational applications can be modified for use on actual projects relatively easily.

Virtually all of the emerging phenomena described in this section cut across and inflict changes in people, process, product, project, and organization application areas. The concepts and boundaries of projects and organizations need to be

rethought when considering trends such as open source and distributed development. People applications become ever more important in light of recent trends such as global, distributed development, more of a user focus, and multicultural concerns. There are process issues with regard to skills, distributed team structures, and culture matching.

Process and product application areas will be impacted by the proliferation of more complex systems of systems, hybrid processes, open source development, distributed global development, and more stringent product demands for higher security and quality. There are increasing needs for COTS, reuse, and legacy systems and software integration. Systems will need to interoperate with each other in ways they have not been designed for.

Project and organization applications will incorporate global collaboration processes. Even the definition of organization must now include distributed open source communities not tied to specific companies, and they have different goals/motivations for performing their work. Projects in open source development do not always have clearly defined beginning and end points.

The interaction of project and organizational dynamics is a complex area for future study. How do product lines, enterprise architectures, and standards interact with each other? The effect of mergers and acquisitions can be profound. Sometimes, organizations change overnight and such events put a big spike into projects and processes.

The new and emerging areas described will have major impacts and thus warrant focused study. They are new enough that substantive system dynamics modeling has not been applied yet, but almost certainly will be in a matter of time. It is expected that these areas will eventually be addressed more thoroughly by modeling and simulation, and some of them will become application areas with example models in the next edition(s) of this book. References are provided to current work in these areas and challenges for future modeling are identified. Additional details on the likely influences of many of these trends over the next couple of decades are in [Boehm 2005a], and trends for systems engineering processes are described in [Boehm 2005b].

7.4.1 Distributed Global Development

Economic trends are disrupting software business models as geographically distributed development processes are becoming ever more pervasive on modern software projects. Software is developed collaboratively in multiple locations around the world, and projects are being contracted out in whole or part for economic leverage. Projects are often split among distributed teams; the teams contribute different portions of work per phase to take advantage of their skill sets and rates. Thus, there is a need for new process models of global, distributed software development

Global development impacts many areas. New types of talent, people, and team-building skills are necessary. People have to collaborate in groups distributed across the globe, requiring new ways of working together with new supporting infrastructure. Processes and products are impacted tremendously. The acquisition of new systems needs to be rethought. These impacts from global development trends are briefly described below.

Strategies for global development cover different areas: product globalization, globally distributed processes, and people issues. These bring new challenges in all areas of products, processes, projects, and organizations. Even open source development is a variation of global development with unique characteristics. See Section 7.4.5, Open Source Software Development.

Global integration also shares characteristics of systems of systems; everything and everybody is increasingly connected to everything and everybody else. Projects are undertaken by multiple teams in different locations, requiring new group processes. Once-standalone systems become parts of large systems of systems they were not designed to interoperate with [Boehm 2005b].

Considerable personnel talent is necessary to succeed with new global opportunities. Distributed group collaboration, continuing education, career path development, and multinational teams provide opportunities for proactive organizations to grow and retain their talent base.

Global outsourcing and teambuilding also requires advanced acquisition management capabilities, including retraining software engineers to think like acquirers rather than like developers (see Section 7.5.2, Related Disciplines and Business Processes). User interfaces for developers, end users, and simulationists need to better reflect the transition from individual to group-oriented performance.

Global connectivity provided by the Internet provides major economies of scale and network-based economies that drive product and process strategies. Developing across multiple time zones may allow very rapid development via three-shift operations, though there are significant challenges in management visibility and control, communication semantics, and building shared values and trust. A groupware mission control concept is an attractive option for distributed process visibility and control (more details are in Section 7.7, Process Mission Control Centers).

Challenges for global collaborative processes include cross-cultural bridging; establishment of common shared vision and trust; contracting mechanisms and incentives; handovers and change synchronization in multitime zone development; and culture-sensitive, collaboration-oriented groupware. Culture matching and localization will be drivers for the new processes and supporting environments [Boehm 2005a].

There is an interesting tie-in between distributed global development and distributed simulation technology. The simulation model topology may resemble the form of the distributed process itself, whereby the nodes of the distributed simulation are coincident with the geographic nodes of the global development. One possibility is that each development site may be responsible for maintaining its own local simulation, and each local simulation may somehow be integrated into a master simulation of the distributed project at large.

An example of using distributed simulation for distributed development is proposed in [Raffo, Setamanit 2005]. This hybrid modeling approach would contain continuous and discrete event submodels. The continuous model portion with system dynamics would have a global submodel for overall project planning and controlling, and several continuous submodels for each development site to reflect differences in human resources and productivity. There would be discrete event submodels for each development site. Each site might have different steps, and artifacts could be passed between sites.

There are also new risks and entanglements with global development. On a multicontractor global project in which different locations maintain their own local simulations, it is inevitable that disputes may arise due to differences in their simulation results and corresponding decision actions. The respective models will be the subjects of evidence for settling controversies.

7.4.2 User- and People-Oriented Focus

> Software is of the people, by the people, and for the people.
> —Barry Boehm and Rich Turner [Boehm, Turner 2004]

As the Lincolnesque quote above reminds us, software is by the people and for the people. They are both software users and producers. The universal proliferation of software and the Internet have made the overall population more computer savvy and, therefore, demanding better user interfaces and quality. For software processes, this begets an increasing emphasis on users and their value considerations.

The user emphasis also entails a global perspective for software used across the Internet and other distributed applications. Software must not only operate in different languages, but it needs to be increasingly culturally aware as well. This implies major interface and design challenges. The issues of culture matching and localization become important. For example, there is a much higher Chinese acceptance level of a workstation desktop organized around people, relations, and knowledge as compared to a Western desktop organized around tools, folders, and documents [Boehm 2005a].

For people issues in a software development environment, see [Acuña et al. 2005] for a process model handbook on how to best incorporate people's capabilities. A focus on people is also a main principle of agile processes, whereby processes are adjusted for individuals and their skillsets. See the next section for this aspect of people-orientation in the process (by the people) versus consideration of people as software consumers (for the people).

7.4.3 Agile and Hybrid Processes

Agile and lightweight processes requiring less rigor need to evaluated quantitatively. The question of scalability is of prime concern. Things are often done loosely without documentation for the sake of speed, and dense people interactions take place. Extreme Programming [Beck 2000] is a good example of a lightweight iterative process that seems to work in limited situations. Two developers share a single workstation and work side by side on the same software. One writes while the other observes, checks syntax, and reviews the work. They periodically switch roles. Even the originators admit that such processes do not scale above about 20 people. Why or why not is this the case? Maybe automated tools should do some of the checking. What dynamic effects occur above certain thresholds of people that impact a lightweight process?

People characteristics and their interaction dynamics are even more important in agile processes. The consideration of personality fit for different roles or processes is

one area of study that spans behavioral sciences and software engineering. We have already seen a major model that accounted for different personality types in Chapter 5—the Burke SPI model that differentiated people by their attitudes toward process improvement. Organizations themselves often have distinct personalities that tend to draw certain types of people and reinforce certain behaviors. But exactly what types of people with what skill sets should companies try to find?

The heavy face-to-face interaction in agile processes is a process within itself that warrants examination. How much close working together can individuals perform? The amount is probably related to individual personality types, which implies that different people might be better suited for different processes.

The degree of change tolerated by people also covers a broad spectrum. Some prefer highly stable environments where they know the project and the organization will not be changing, whereas others enjoy the rapid dynamics typified by fast-paced Internet projects in which the business goals and development environment often change. Companies will always deal with finding the right balance between flexibility and discipline. Like crossing a chasm, they cannot lean too far on either side. And the balance shifts over times with market and other business trends.

Hybrid processes are emerging in several forms. On distributed, global projects they may be comprised of multiple local processes in conjunction with integrated processes with combined teams. Some projects may be combinations of open-source and closed-source development. On other large complex projects, they might be manifested as combined agile and disciplined processes. Hybrid processes may be composed of "hybrid" teams of different types of people skills, as a variant of Conway's Law.

For example, on large and complex systems-of-systems projects, strategies are being developed for balancing agility and discipline. Separate teams of people with different skills can be used to cope with rapid change while retaining project stability. Areas of faster product change are dealt with by separate teams of "agile" people equipped for assessing rapid change, whereas other more "disciplined" teams focus on stabilizing software baselines and plans amidst the changes. Development can be stabilized using short cycles and the right people on the right teams. Change can be foreseen, assessed, and built into the product architecture using as short increments as possible [Madachy et al. 2006].

Some people thrive on order and others thrive on chaos. These differences can be utilized for different aspects of a project to rebalance it when necessary. Agile-oriented people are rapid learners and good at assessing new technologies and COTS in light of unplanned changes. Plan-driven teams can develop the specifications in a more disciplined fashion. On larger projects, another team might consist of dedicated people who are continuously doing verification and validation (V&V). They would be good at critiquing but not necessarily very agile.

At USC, we are currently modeling hybrid processes for an extended spiral life cycle to address new challenges for software-intensive systems of systems (SISOS), such as coping with rapid change while simultaneously assuring high dependability [Madachy et al. 2006]. See the model highlighted in Chapter 4 for optimizing the number of agile people in a hybrid process.

Another recent effort at modeling agile processes is described in [Fernández-Ramil et al. 2005]. The authors use qualitative modeling to investigate agile processes after having success with qualitative modeling of open source community processes per [Smith et al. 2005].

7.4.4 Commercial Off-the-Shelf Software

COTS was briefly introduced and defined in Chapter 5, which highlighted a model for COTS glue code development and described a model concept for a COTS life-span model. However, those just addressed a very small part of the totality of COTS issues, which continue to change. This section will expand on the Chapter 5 introduction to address current and future trends of COTS.

The trend of building systems incorporating preexisting software is one of the most significant changes in software development. However, COTS products are examples of a disruptive technology with major potential advantages and major but poorly understood impacts on current practices. The use of COTS products generally introduces large, opaque, uncontrollable components with large multidimensional sources of advantages and disadvantages into a software application's solution space, causing fundamental changes in the nature of software requirements engineering, design, development, verification, evolution processes, methods, and economics.

To provide a focus on the types of applications for which COTS considerations significantly affect the dynamics of the development process, a COTS-based application (CBA) is defined as a system for which at least 30% of the end-user functionality (in terms of functional elements: inputs, outputs, queries, external interfaces, internal files) is provided by COTS products, and at least 10% of the development effort is devoted to COTS considerations. The numbers are approximate behavioral CBA boundaries observed in COTS-based system (CBS) projects [Boehm et al. 2003].

At USC, we have identified three primary sources of CBA project effort over years of iteratively defining, developing, gathering project data for, and calibrating the Constructive COTS (COCOTS) cost estimation model [Abts 2003]. These are defined as follows:

- *COTS assessment* is the activity whereby COTS products are evaluated and selected as viable components for a user application. This includes searching for suitable products, evaluating them with respect to the needs of the project and the capabilities of the developers, and choosing the best-fit candidate COTS products.
- *COTS tailoring* is the activity whereby COTS software products are configured for use in a specific context.
- *COTS glue code development and integration* is the activity whereby code is designed, developed, and used to ensure that COTS products satisfactorily interoperate in support of the user application. Glue code, also called "glueware" or "binding" code, is the code needed to get a COTS product integrated into a larger system (see the Chapter 5 example on glue code).

Other sources of effort not covered in COCOTS include:

- *COTS volatility* effort is that required to adapt to COTS product changes over time, including minor version updates and major new releases.
- *CBS evolution* refers to the ongoing modification, adaptation, and enhancement of a COTS-based system after its initial deployment. Sometimes, this effort can be quite substantial.

An important consideration is that there is no one-size-fits-all CBS process model. Assessment activities, tailoring activities, and glue code development are not necessarily sequential. Many different ordering and iteration patterns of these activities have been observed [Boehm et al. 2003, Port, Yang 2004]. Such differences in the activity sequences result from specific project characteristics and associated risks.

There has been also been a wide variation in the observed effort distribution of CBS assessment, tailoring, glue code development, and maintenance across projects [Boehm et al. 2003]. The assessment and tailoring efforts vary significantly by the class of COTS products (e.g., OS, DBMS, GUI, device driver, disk arrays, compilers, word processors).

CBS post-deployment costs may significantly exceed CBS development costs, and rise faster than linearly with the number of COTS products integrated. Although there is some anecdotal evidence for this, the greater than linear relationship is strongly suggested by the fact that n components will require on the order of n^2 interfaces to be considered, and the effort in developing and maintaining COTS-based systems generally lies in those interfaces. The implication is that short-term planning that often plagues new development is even more risky in CBS development projects. A summary of lessons learned from COTS system maintenance is presented in [Reifer et al. 2003].

New risks due to COTS products include:

- Having no visibility into COTS internals
- No control over COTS evolution
- Unpredictable interactions among independently developed COTS products

For such risks, traditional programming-oriented software engineering guidelines are occasionally useful, frequently irrelevant, and often dangerous. Examples of the latter are traditional approaches to software requirements, object-oriented design and development, and complementary black-box/white-box testing.

Traditional sequential requirements–design–code–test (waterfall) processes do not work for CBS [Benguria et al. 2002], simply because the design decision to use a COTS product constitutes acceptance of many, if not most, of the requirements that led to the product and to its design and implementation. Usually, a COTS product's capabilities will drive the "required" feature set for the new product rather than the other way around, though the choice of COTS products to be used should be driven by the new project's initial set of "objectives" rather than "requirements." Additionally, the

volatility of COTS products [Basili, Boehm 2001] introduces some recursion and concurrency into CBS processes.

Glue code and overall CBS costs and risks can be reduced by investments in COTS assessment. The implication is that developing glue code or other CBS activities without sufficient COTS assessment may incur unexpected critical risks. Risk reducing assessment approaches include benchmarking, prototyping, trial use, and reference checking. COTS assessment is another source of effort for which "how much is enough?" can be answered via risk analysis. This involves balancing the risk exposure of doing too little COTS evaluation (picking the wrong combination of COTS products) with the risk of doing too much COTS evaluation (incurring too much delay).

The average COTS software product undergoes a new release every 8–9 months, and has active vendor support for only its latest several releases. Survey data supports the 8–9 months release cycle, whereas the evidence for the number of supported releases is anecdotal and varies widely. The implications are high adaptation costs not only during maintenance, but also during development and transition, especially if the CBS evolves in a different direction than the underlying COTS components.

There are a number of important research issues regarding the dynamic aspects of CBS development and evolution. These include adaptation to COTS releases, COTS refresh strategy analysis, balancing COTS risks, process concurrence, glue-code dynamics, tailoring, and more.

Software systems that contain multiple COTS packages often need to synchronize releases from several vendors with different update schedules. The challenge is timing the product "refresh" as new COTS releases are put out. The problem is exacerbated in complex systems of systems with distributed platforms. How frequently the components should be "refreshed" and the overall software rebuilt is a complex, dynamic decision.

Models describing the interrelated factors and feedback in complex CBS and evolution processes can help in decision making and process improvement. For example, the effects of interactions between COTS filtering, detailed package assessment, tailoring, glue-code development, integration with legacy systems and other applications, system testing, and evolution can be quantified and better understood. Users can assess different "what-if" scenarios and plan CBS projects with better information. Improved CBS processes can be defined from this.

Open-source development is a related topic that has much in common with developing systems with COTS. However, there are some unique differences. The open-source phenomenon is gaining a lot of ground but currently there are no modeling applications using system dynamics. This new and emerging area is discussed next.

7.4.5 Open Source Software Development

The phenomenon of open source software development has gained significant prominence in the last decade. There is still much to understand about the different people factors, group dynamics, quality attributes, overlapping project dependencies, different evolution dynamics, and other phenomena compared to more traditional closed source or COTS development. Open source is also a variant of global development, with

many of the same considerations mentioned in Section 7.4.1, Distributed Global Development.

Open source development also has much in common with COTS-based systems, except for some unique characteristics. The source code is by definition available and transparent to all. Anyone can potentially have an influence on new features or participate in other aspects of development, so the processes used in the open source community present unique social interaction models.

Many point out the successes of the open source Linux operating system and other projects in terms of increased security, reliability and stability when compared to their closed source contemporaries. But some security experts are skeptical about the ability to assure the secure performance of a product developed by volunteers with open access to the source code. Proliferation of versions can also be a problem with volunteer developers. Open source may not be optimal for all classes of software. Feature prioritization by performers is generally viable for infrastructure software, but less so for corporate applications software [Boehm 2005a].

The usage and reach of the Internet as a universal communicational medium has helped fuel the recent tremendous growth in the open source movement. Traditionally, the communication overhead and complexity of a project increases by the square of the number of people. However, in open source development processes with thousands of programmers, the effects of onerous communication overhead are mitigated through the extensive use of a central versioning control system, a small core development group, a large set of beta testers, and frequent releases.

Quality dynamics are different in open source systems, partly because "Many eyes make all bugs shallow" (also called "Linus' Law) [Raymond 2004]. With so many open source testers, a high percentage of defects are found in every release. With frequent releases, updated and increasingly reliable software gets to the end user quicker. Open source development requires a critical mass of beta testers for this speedy feedback.

The success of open source development is highly dependent on the formation and evolution of their supporting communities, which may include end users, beta testers, developers, and others. Communities provide a sense of belonging and rapid feedback, and are why developers stay on projects [Jensen 2004]. Core developers listen closely to these communities and try to incorporate the suggested changes.

Group dynamics play a big role in open source development. Modeling the process factors of community peer review and collective decision making would enable better understanding of open source methodologies. These are also common concerns with (closed source) globally distributed development.

Few open source projects are started and built from scratch, and reuse is typically high. Usually, there is the presence of a critical starting mass of software elements, which themselves have to be open source in order to allow modifications. There are also aspects of product lines to consider. A large number of open source projects become dependent on one another, much due to the philosophy of code sharing. The interdependencies may be at different granular levels and have been studied in [Scacchi 2004a, Schach 2002, Smith et al. 2005].

People dynamics and their motivation are essential areas of study for open source development processes as the factors vary greatly when compared to traditional COTS

processes [Scacchi 2004b]. Why do people come together and persist during the development of an open source product? Surveys have revealed that open source projects are viewed as a venue for learning, positions are based on interest, and a high percentage of participants contribute to multiple open source projects [Hann 2002, Hertel 2003]. Primary motivational factors are a sense of betterment of one's skills and self-satisfaction. A large majority of open source contributors develop a personal stake in their projects and many work tirelessly.

Web-based open source software development project communities provide interesting and unique opportunities for modeling and simulation of the process and people dynamics. Much of the process data is recorded and public, such as bug databases and code repository histories. This empirical data can be leveraged by researchers. For example [Jensen, Scacchi 2005] focuses on processes both within and across three distinct but related open source project communities: Mozilla, the Apache HTTP server, and NetBeans. They looked at the process relationships within and between these communities as components of a Web information infrastructure. This modeling helps to understand the open source development processes utilized by their respective communities and the collective infrastructure in creating them.

Open source systems may not neatly follow the well-known laws of software evolution in [Lehman 1980, Lehman 2002]. Recent modeling of open source software evolution is described in [Smith et al. 2005], where qualitative simulation is used to examine the general behavior of models of software evolution. This approach to the empirical study of software evolution system dynamics was used because neither the empirical data nor precise relationships were all available.

The qualitative modeling of open source growth trends and other empirical data in [Smith et al. 2005] looked at drivers of software evolution. The authors report on the application to data from 25 open source software systems. They compared model output with qualitatively abstracted growth trends, and looked at trends of functional size and complexity to those predicted by the models. The results suggest that the study of the relationship between size and complexity and its interaction via stakeholder feedback loops has a role in explaining the long-term evolutionary behavior of open source systems. This same qualitative approach is being extended for the evaluation of agile processes [Fernández-Ramil et al. 2005].

The same authors have applied an agent-based approach to studying open source processes. In [Smith et al. 2006], they model both users and developers in the evolution of open source software. They included factors for productivity limitations, software fitness for purpose, developer motivation, and the role of users. They compared the results against four measures of software evolution and found good fidelity with the model.

7.4.6 Personnel Talent Supply and Demand

There are global and national software skill concerns that warrant more rigorous examination. System dynamics has a long history of being applied at very high macro levels of people, such as an ongoing world model of population and resource dynamics [Forrester 1973]. One issue of prime importance for the software industry is the disparity

between supply and demand for software engineers. The level of industrial demand versus the number of students being educated with requisite skills should be quantified for long-term policy making. All indications are that the growing demand for software will outstrip the supply in the foreseeable future.

With demands for software and information technology growing at a blinding pace, it is no surprise that there is a shortage of qualified personnel. A previous report [DoD 2000] suggested that there were more than 50,000 unfilled software jobs in the United States alone and that the number was growing at a rate of more than 45,000 jobs per year. Other sources indicate the current shortage as high as 400,000 jobs [DoD 2000]. This gap will continue to put pressure on the industrial base to maintain adequate staffing levels.

The shortages are industry-specific and caused by economic fluctuations. For example, aerospace and defense companies in the United States today typically have very highly experienced people. However, their workforce will be losing critical domain knowledge in the next two decades as these people retire. The tremendous growth of commercial software work has tipped the balance to attract the vast majority of entry level and early career software professionals away from aerospace and defense companies.

Witness also the impressive growth of software companies in India and China in recent years. They have produced some top-notch software development and consulting firms that can severely undercut United States labor rates. The impact to established consulting firms is being felt in terms of lost contracts.

In the discussion on agile and hybrid processes, the existence of radically different types of personalities was noted, as was the need to capitalize on their differences. Some people like fast-changing conditions, whereas others prefer stable environments. Some enjoy interacting with many people and some prefer to work alone in a corner. In today's world, with hybrid processes and so many different stakeholders to deal with, there is room for all types of people in software development and evolution.

7.5 MODEL INTEGRATION

7.5.1 Common Unified Models

The nature of process modeling is that different process concerns drive different factors to be included in particular models, but the concept of a unified model is attractive nonetheless. Different models can be integrated through combination and portion replacements. There are several candidate foundation models such as the Abdel-Hamid project model, but it is unlikely that a consensus set of models can be reconciled. Similarly, an attempt to combine the best of all software cost models is bound to fail just due to differing base assumptions. Business and market forces alone will keep models proprietary and insular, since in-house models use sensitive local data and sales of modeling products depend on having differentiating factors from competitive offerings.

Models for different life-cycle processes could be integrated in a unified system dynamics model for comparing multiple alternative lifecycle processes. It would be used

to help select appropriate life-cycle process model(s) from the myriad of choices. An example of a life-cycle process model decision table can be found in [Boehm 1989], though it needs to be updated for modern practices.

An interchange standard for system dynamics models would go far to promote unification of different models. XML is a good candidate as a viable and natural solution for system dynamics model interchange. Details of a proposed approach for an XML-based standard are in [Diker, Allen 2005].

7.5.2 Related Disciplines and Business Processes

In most circumstances, the software process is one of several processes to be integrated in an enterprise. Businesses will be embedding simulation into larger enterprise-wide applications that encompass many disciplines. Some examples include systems engineering, business processes (e.g., sales, hardware development, supply chain, etc.), or acquisition processes undertaken by large governmental agencies. As described in the book *Serious Play* [Schrage 2000], organizations will increasingly continue to resort to heavy use of simulation to gain competitive advantage.

A prevalent trend is the increasing integration of software engineering and systems engineering disciplines. This is reflected in the CMM-I, superseding the Software CMM, the refocusing and expansion of many organizations to recognize and integrate both disciplines (even going so far as renaming their companies), and major conferences like the Systems and Software Technology Conference now incorporating both disciplines. There have not been any system dynamics models to date that exclusively focus on their integration dynamics. Process concurrence relationships between systems and software work phases could be applicable.

Integrated models will include more disciplines and business aspects, such as integrated systems and software engineering processes, business processes with software processes, and integrated views of acquisition processes and supplier development processes on large government projects with acquisition oversight. Simulation will also be integrated with more traditional project management tools. These include planning and scheduling tools, earned value reporting systems, and so on.

Simulation to support business case development was highlighted in some Chapter 6 applications. Software business value analysis is a crucial area that simulation will increasingly address. Business case simulation requires industry-specific knowledge to analyze particular software product markets and their associated business processes. Conversely, business considerations are a major external driver of software processes.

An integrated acquisition and software process model would involve modeling overall system feasibility as well as process development considerations (e.g., cost, schedule, quality). A virtual acquisition process was described in the SAMSA proposal [Boehm, Scacchi 1996], where system dynamics was incorporated for project modeling. A system dynamics model framework for a software supplier acquisition process for the automobile domain was described in [Haberlein 2004]. More recently, the acquisition process for a large-scale government system has been the focus of a multiyear research project at the Aerospace Corporation [Abelson et al. 2004, Greer et al. 2005].

[Greer et al. 2005] describes the results to date by the Aerospace Corporation for the use of simulation to better understand the software-intensive system acquisition process. A case-study approach is used to explore the dynamics of "disconnects" in baselines across multiple organizations in a large software-intensive space system development program. Disconnects are the latent differences in understanding among groups or individuals that can negatively affect the program cost, schedule, performance, and quality should they remain undetected or unresolved.

In [Greer et al. 2005] the authors have constructed a system dynamics model of communication effectiveness and delay across four organizations that sequentially and iteratively rely on each other for requirements and deliverables. Their analyses from multiple simulations suggest that the highest points of leverage in reducing disconnects are in increasing expertise levels, improving communication clarity, and accelerating the pace of assessing the impacts of changes in partner organizations' understandings and actions. These results oppose traditional assumptions that disconnects are due to external requirements changes and that speeding up organizational processes will reduce disconnects.

Chapters 3 and 6 provides some stock models of integrated systems engineering, hardware engineering, marketing, sales, and manufacturing processes. These may serve as starting points for integrated enterprise process models.

7.5.3 Meta-Model Integration

In this section, *model* is used in a meta sense; it refers to a general modeling technique or perspective and not specific instances like the application models.* Results in any scientific or engineering field are stronger when validated using different perspectives or models. A related best practice in software estimation we have been preaching for years is to estimate using several methods [Boehm 1981], compare the results, and hone in on a better result. If different approaches provide disparate results, then evaluate their respective assumptions to find discrepancies and iterate. Generally, the estimates converge and more confidence can be had in the results.

Likewise, system dynamics presents one way of viewing the world and it is best used to balance and complement with other modeling techniques. Meta-model integration may refer to the following types of models:

- Static and dynamic models
- Continuous, discrete, agent-based, and combined models
- System dynamics and analytic dynamic models (e.g., analytic Rayleigh staffing model)
- Quantitative and qualitative models
- Mathematical and heuristic models
- System dynamics in support of theory validation in general (see Section 7.6)

*The term meta-model refers to any type of model representation, as opposed to "metamodels" from [Barros et al. 2006b], which are system dynamics models specific to software processes.

Integration may take on several forms. It can include identifying model differences and similarities in assumptions or constructs, combining models that may or may not have common elements, taking selected parts of different models to create new ones, comparing and interpreting results between models, calibrating between models, or even simplifying models in light of each other.

One example of a modeling technique "filling the gaps" of knowledge for another is qualitative simulation in conjunction with system dynamics. If the system dynamics of a process are under investigation but empirical data and precise functional relationships are lacking, then a qualitative model can at least provide output results with general shapes and trajectories.

7.6 EMPIRICAL RESEARCH AND THEORY BUILDING

Simulation will increasingly be used to support empirical research in software engineering, and vice-versa, as simulation becomes more widely accepted and practiced. There are several ways in which simulation and empirical studies can be used together synergistically. Simulation models can support real experiments to strengthen other empirical methods and be used for evaluating theories [Munch et al. 2003, Rus et al. 2003]. Or empirical results can be used to develop better models. [Munch, Armbrust 2003] is an example of using empirical knowledge from replicated experiments for simulation.

Simulation can provide a virtual laboratory for low-cost experimentation compared to live experiments or field studies that require a lot of resources. Simulation is an efficient and powerful way to test theories this way. Empirical observations required for nonsimulation approaches may be very difficult and time-consuming to generate and assemble, and, thus, simulation can help accelerate progress in empirical research. When the empirical data is virtually impossible to obtain, such as trying to conduct a large industrial project more than one way to compare techniques, simulation may provide results when traditional methods cannot.

Simulation models can be calibrated with a relatively small amount of empirical data or expert-determined values. The scope of the models can potentially be broadened to include other factors that have been empirically determined or with expert judgment when empirical results are not available. Then experiments can be run without the need for actual projects or other expensive data collection. Sometimes, the models will also help identify new measurements that are necessary to fill gaps, so simulation may, conversely, help refocus empirical data collection.

In the traditional process of theory building, a problem is identified, a hypothesis is formulated, experiments are defined to test the hypothesis, resources are gathered for the experiments, and, finally, the experiments are executed and analyzed with respect to the hypotheses. Ideally, the experiments can also be independently replicated. Simulation provides an alternative way to prepare for and run the experiments, and the results of the simulations can be used for analysis.

Proposed theories can be incorporated and tested in a simulation framework, incrementally added to, or alternative theories can be substituted. By changing simulation parameters, each theory can be tested individually or combinations of noncompeting theo-

ries can be evaluated. If any fragments of relevant data do exist, they can be compared to the simulation results. As in other fields, a simulation framework can incrementally incorporate newly accepted theories to serve as a unified model and experimental testbed. The framework can potentially integrate both macro and micro process views as well.

Frequently, there are complications in the real world so experiments need to be iterated more than once. Sometimes, the results of live experiments indicate a need for revisions in the experimental design. As in all scientific studies, empirical research must also be repeatable by others to validate new theories. Simulation makes it much easier to rerun experiments for all these reasons. Independent researchers trying to replicate results can also resort to their own independent simulation models to validate the results of others. The practice of using alternative simulation models in parallel to test theories about complex phenomena is common in other disciplines.

The research in [Raffo et al. 1999a] identified the scarcity of empirical studies relating to the practical impact of process modeling and simulation. The authors addressed some relevant aspects of the multifaceted relationship between empirical studies and the building, deployment, and usage of software process models. They identified empirical issues encountered when trying to analyze the following:

- Process data used as direct input to simulation models and/or used to assist model building
- Model output data used to support management decisions about process alternatives
- Model structure in the context of evaluating the efficiency of a process
- The effectiveness of process models in supporting process change and improvement

Work needs to be done on techniques for empirical analysis of model inputs, analysis of simulation outputs, empirical evaluation of process models, and the empirical evaluation of process models in supporting process improvement. A recurrent theme is the complementary nature of modeling and measurement [Raffo et al. 1999a]. Simulation models provide a framework and focus for measurement programs.

7.6.1 Empirical Data Collection for Simulation Models

Simulation does not eliminate the need for empirical data collection, so there is still substantial future work to be done in this area. Progress in virtually all of the modeling application areas depends on real-world data for inspiration, analysis, and other usage in the models. Empirical data from real projects and experiments still needs to be continuously collected and analyzed. Accurate and consistent-quality data is desired and therein lays the challenge as in all software metrics activities.

Empirical data can be highly valuable for software process simulation studies, yet there are fundamental challenges in its acquisition and usage. Data is used in calibration and the driving of and validation of simulation models. When simulation is used for decision support, empirical data is used to describe the distributions for different

decision options. Research results ideally can be validated by others with different sets of data from multiple sources.

Some potential problems to overcome when acquiring data for simulation models include:

- Large effort in generation, collection, and possible reformatting of data for simulation models
- Effort to locate and assess relevance of empirical data from external sources
- Existing data may not be commensurate with model constructs in terms of granularity, abstraction level, and so on
- There are so many different processes
- Field data does not represent hypothetical future processes
- The lack of data may constrain the scope of process models

The use of experimental data or observational field data for simulation depends on the goals and constraints. It is frequently desired to bolster models with data to reflect reality as closely as possible, but detailed data is not always warranted. Conversely if it is not available, its lack may become a constraint on the simulation study. As discussed in Chapter 2, a balance must be struck because such data rarely exists in perfect form, if at all. Some metrics frameworks to address these issues were described in Chapter 2 in Section 2.11, Software Metrics Considerations, including GQM, IMMoS, and the WinWin Spiral life cycle as a risk-driven approach to simulation studies.

Initiatives can also be undertaken in the larger software community to align diverse efforts for traditional experimentation and process simulation modeling. Such initiatives would define and implement research approaches that combine empirical measurement and process simulation. Researchers could work more closely with industry to obtain empirical data for their research. New techniques could be developed to transform data or deal with the lack of data. Improving access to and interpretation of existing empirical data and process models would also be a great advantage to the community.

Whereas organizations maintain local empirical data caches (to differing extents), there are no freely available public domain sources of comprehensive empirical data to validate research studies at large. Open source repositories of empirical data are one possible way to help alleviate the lack of data. For example, on a CeBASE research project sponsored by NASA we developed an online experience base for researchers on best practices related to high-dependability computing [Boehm et al. 2004]. Contextual interpretations were provided with the raw data. A long-range option is to extend this type of work and provide public sources of empirical software process data.

7.7 PROCESS MISSION CONTROL CENTERS, ANALYSIS, AND TRAINING FACILITIES

Simulation will become part of the process itself and a natural way of doing business. People will be using models much more frequently to help determine next steps and re-

plans in response to changes. Project dashboards that provide "the big picture" are currently used as vehicles for management insight and project control, and they will evolve to include more models as views into projects. Risk mitigation and replanning will frequently involve a change of model(s) for all to see.

Measurement-driven dashboards provide a unifying mechanism for understanding, evaluating, and predicting the development, management, and economics of systems and processes [Selby 2005]. Dashboards enable interactive graphical displays of complex information and support flexible analytic capabilities for user customizability and extensibility. The displays help visualize data more intuitively, identify outliers, and support drill-down capabilities. Dashboards have been used on actual large-scale projects and empirical relationships have been revealed by the dashboards. In [Selby 2005], the dashboards revealed insights on leading indicators for requirements and design of some large-scale systems for feedback to the organization.

Several of the simulation applications in this book have demonstrated dashboard-like displays using top-level graphical interfaces with interactive gauges, sliders, input boxes, and output displays. The dashboard concept can be substantially extended by incorporating more models and interactive simulations, plugging into actual process data to drive the models, and allowing group interaction.

We will define a process "mission control center" as an extended dashboard concept employing multiple adaptive models with real-time data streams. It provides a model-pervasive, multidimensional view of processes. The model outputs are shown with past actual process data and extrapolated into the future. With multiple models that may indicate different outputs, decision makers can consider the context and applicability of the models in specific situations. Models can be deleted over time when they fall out of favor or are updated for improvements.

This usage of simulation brings it more into the fold for "control and operational management," which is one of the purposes of simulation identified in Chapter 1 but has very few examples. It goes much further than planning or tracking, by enabling users to make real-time control decisions using simulation results and actual project measurements.

The mission control center metaphor goes beyond traditional dashboards because it not only supports project tracking but, like space mission control centers, it supports preparations for major changes and unanticipated events through the models. "What ifs" can be conducted at any time, leveraging the actual process data, and models can be recalibrated or otherwise updated in real time as necessary. The data streams may be of different periodic rates, whether weekly and monthly metrics traditionally used for project tracking [Baumert et al. 1992, McGarry et al. 2002] or per minute telemetry such as provided by the Hackystat tool [Johnson et al. 2005].

During space missions, control centers are manned 24 hours each day, and such continuous monitoring would be relevant for large 24/7, globally distributed projects as multiple teams hand off work. Real-time trends of geographically distant teams would be instantly recognizable at all mission view centers.

As a model-intensive dashboard, the mission control center employs multiple simulations with possibly competing or contradictory models, allowing comparisons of alternative simulation approaches to help decide when one approach is more suitable.

Some may go out of use when they become irrelevant, events cause the assumptions of the model to be wrong, or they are judged too defective for repair. The models adapt over time as the "mission" unfolds. Behind the scenes are model learners to aid in the continuous model updating and deleting.

Suppose one aspect or portion of the mission control center is dedicated to defect trends. There will be arrays of continuous model outputs plotted against actuals and arrays of discrete mode results plotted against actuals. These kinds of simulations will be used to test changing assumptions. Some of the models will be recalibrated at some point and reflect their updated simulation results. Some models will be better predictors then others, possibly at different phases. Some of the models will be deleted or retired when they become irrelevant or invalid.

The control centers also have access to virtual reality simulations with humans in the loop to practice or test how people will react in a given situation. For example, a process performer can be inserted into a virtual environment like SimSE [Oh Navarro, van der Hoek 2005] that takes on the characteristics of the actual project. Or a manager can propose that a value-based decision be made in a SimVBSE environment [Jain, Boehm 2005]. The participant can use actual process measurements and simulations of his choice to help assess the value trade-offs and support his decision. The virtual environment can even include internal simulations for observation and manipulation.

All of the aforementioned elements of a process mission control center would share a common database of past and actual process data, and there would be conceptual links between the different model representations. For example, the individual discrete entities in a discrete event model would be the same ones aggregated in a continuous representation. The user view would depend on the modeling paradigm used and chosen level of aggregation.

7.8 CHAPTER 7 SUMMARY

Simulation usage will grow in diverse disciplines to extend knowledge, and software engineering should not be an exception. Improvements in simulation technology and underlying computing infrastructure will help fuel dramatic advances. Such advances are already happening in many fields and, likewise, software process simulation will continue to become more immersive and extensively used. But we also need to improve the practice of software process simulation to cope with perpetual rapid change.

Simulation will become the way to do business in the future. Simulation systems will often be used as part of larger decision support and enterprise systems, and major decisions will be assessed using simulations that encompass all aspects of business operations. Simulation software will become more open to allow integration with related applications. Simulation will be widely used, but people will be using more simulators specifically designed for their business processes instead of using generic simulation tools.

Many of the future thrusts described in this chapter are connected and will contribute to each other. For example, better modeling tools and methods will help simula-

tion become more widely accepted and easy to use for software processes. As simulation becomes more accepted and usable, it will be applied more frequently in empirically based research. Seeing the fruits of simulation-based research will help spawn even better tools and methods, with increased usage by the practitioner community. The tools and methods include component-based modeling, automation of model building and output analysis, statistical support and experimental design, more interactive and lifelike simulations, enhanced project dashboards with simulation, game playing and training applications, and so on.

Simulation software of the future will provide enhancements for model creation, input data analysis, output data analysis, and decision making. Novel techniques such as machine learning or genetic algorithms will be used to better understand or update models. Improved graphics and virtual reality can make the environments more appealing and powerful. Environments will be customized for specific applications such as software processes, but also allow integration with other enterprise aspects and disciplines for modeling more comprehensive systems.

Model building will become more automated and domain-specific. Tool advancements such as component-based or automated model construction kits can make simulation resources more easily available to people. Tools will mature to optimize the creation of software process models specifically, and those models may be integrated with models developed with other tools optimized for their respective domains.

Object-oriented technologies have become pervasive in software development and will also be used more in simulation. Models will be created with object-oriented components and sometimes be driven by object-oriented databases, depending on the application. Techniques such as system dynamics meta-modeling will be used for construction and reuse of domain models.

In the future, simulation software will no longer be so cleanly divided among continuous, discrete event, or agent-based methods. Applications will incorporate hybrid modeling to capture different perspectives and allow multiple-view insights. The different methods can all share common data and be linked through intermodel relationships in many cases.

New application models will cover related trends such as globalization and distributed development, open source development, increased emphasis on the user and people concerns, new agile and hybrid processes, increased product demands with more complex systems, and personnel shortfalls to meet the new demands. These areas are additive to the types of applications already showcased in Chapters 4–6, and will be viewed from new perspectives such as value-based software engineering.

These application trends demonstrate increasing connectedness of people, processes, products, projects, and organizations. Similarly, there will be more connected models of all these aspects. Multidisciplinary teams will use integrated sets of simulators. Enterprises will continue to integrate simulations of all aspects of their operations. However, the traditional boundaries of projects and organizations will sometimes be redefined in light of new trends. Software process models will be components of more increasingly complex and integrated simulations encompassing more business aspects.

Many of the directions in this chapter have multiple new aspects to them, such as current efforts using both new modeling techniques and modeling new application

areas of software processes. This belies the fact that rapid change is also occurring in software process modeling and simulation.

Simulation will strengthen and streamline many aspects of empirical research and theory building. When combined with empirical methods, simulation is a powerful way to test theories and extend knowledge in the field (partially aided by new simulation tools for experimental design). Some expensive and time-consuming empirical investigations can be reduced or totally eliminated through the use of simulation. This different mode of research is expected to provide new insights at a faster pace than in the past. This has already happened in other disciplines, and the only barriers in this field are more widespread acceptance and development of simulation skills/resources.

Extended dashboards and mission control centers will comprise many of the aforementioned concepts. They will be model-intensive interactive displays that are used in all aspects of process operations. Models will be phased out continuously, sometimes due to improved models (aided by model learners behind the scenes) or because the phase of a project or enterprise changes. As the real-life systems change, then corresponding simulation models need updating. For instance, when a software system goes into operation, its development models will be replaced or enhanced with models for its evolution.

This book proudly carries on the tradition of past pioneers such as Forrester, Abdel-Hamid, and others. It is hoped that this book will help guide others in their quest to have successful projects make useful software for people, through better understanding and informed decision making in complex scenarios.

We will have to change our ways of thinking because simulation will indeed be more pervasive in the future. As we practice simulation, it will induce even more changes about how we view and architect software processes. The models will change us, not just vice-versa. Modeling applications will be more sophisticated, with more impact on the bottom line and affecting the daily activities of many software professionals. It is time for serious play, so rev up your simulation engines.

7.9 EXERCISES

7.1. Choose one or more of the models provided in this book and improve their interfaces so that people can use them easier. Add control panel capabilities to make them more user friendly. Develop appropriate scenarios to run them interactively. If you have access to advanced rendering technology, you may investigate transforming them to be more graphical or potentially incorporate virtual reality.

7.2. Discuss issues related to groups of people accessing, modifying, and running common simulation models. Consider interface, configuration management, and other issues impacted by group usage.

7.3. Choose one or more of the models provided in this book and adapt them to run on the Internet or other network. Furthermore, adapt them for collaborative distributed usage across the network by multiple people.

7.4. Conduct research and write an essay on how system dynamics tools can be integrated with expert system shells. Note that several entries in the annotated bibliography address expert systems.

7.5. Apply machine learning techniques on a model chosen from this book or another source, or create another model and analyze it. Describe what was learned from the model and implications for model improvement.

7.6. Create a mockup of a process model construction utility for any chosen aspect(s) of software processes. You can focus on particular aspects such as effort or defect modeling, or be more comprehensive. You may also focus on particular applications such as product line modeling, global development, and so on. Develop use cases that describe the interaction with such a utility.

7.7. Investigate the integration of systems and software engineering processes. Create a model with policy levers to represent options on how the integration might work. Possible issues to analyze are having separate or conjoined processes, the timing of artifact handoffs (process concurrence might be handy), the effect of (non)participation of software engineering personnel on developing requirements, process serialization versus iterative parallel activities, and so on.

7.8. Assess and critique some of the stock models provided in Chapter 6 for integrated enterprise processes. If feasible, adapt them for software development scenarios for a chosen environment.

7.9. An older example of an application that embodies some of the ideas in this chapter was described in [Boehm, Scacchi 1996]. Tom DeMarco developed the hypothetical project simulation tool in Appendix 2 that provides the following abilities to multiple stakeholders (acquirer, user, developer, etc.):

- Simulating project outcomes under different what-if conditions
- Simulating different risk conditions to give managers a chance to work out reasonable responses
- Simulating later stages of project work from a base of actual performance to date (the model is updated with actual data as the project continues)
- Creating a common metaphor to allow program managers to interact fruitfully with their peers

A simulation scenario and narrated experience of interacting with the tool is also described, whereby a user iteratively hones in on a suitable project scenario balanced by risk. He eventually locks in the required capability and specifies constraints until the risk is acceptable. Update this concept for other features of advanced simulation environments such as distributed usage, among others. Go further and develop a prototype or a working tool based on these concepts.

7.10. Choose any of the new and emerging application areas and develop a simulation research project around it.

7.11. Examine the future supply versus demand of software personnel for a chosen region (e.g., United States, India, the world). Identify important dynamic fac-

tors, indicators, and needs of the employment market. Analyze the experience levels and the number of well-trained people based on actual data. Project the future supply and represent the trend(s) in a system dynamics model. Extrapolate out 5 and 10 years into the future. What are the consequences? Are there any national or international policies to recommend?

7.12. Create a mockup of a process mission control center showing multiple models with updates and retirements. It can focus on a particular aspect and does not have to contain every item mentioned. Create storyboard(s) that describe the scenario of how the control center is used and what is being displayed. This may include project history to explain trends and model changes.