SOFTWARE PROCESS DYNAMICS

Raymond J. Madachy

IEEE PRESS

WILEY-INTERSCIENCE
A JOHN WILEY & SONS, INC., PUBLICATION

SOFTWARE PROCESS DYNAMICS

IEEE Press
445 Hoes Lane
Piscataway, NJ 08854

IEEE Press Editorial Board
Mohamed E. El-Hawary, *Editor in Chief*

R. Abari	T. G. Croda	R. J. Herrick
S. Basu	S. Farshchi	S. V. Kartalopoulos
A. Chatterjee	B. M. Hammerli	M. S. Newman
T. Chen		

Kenneth Moore, *Director of IEEE Book and Information Services (BIS)*
Catherine Faduska, *Senior Acquisitions Editor*
Jeanne Audino, *Project Editor*

Technical Reviewers
Raman Aravamudham, University of Iowa
Márcio Barros, UNIRIO/Brazil
Guilherme H. Travassos, COPPE/Federal University of Rio de Janeiro, Brazil

SOFTWARE PROCESS DYNAMICS

Raymond J. Madachy

IEEE PRESS

WILEY-INTERSCIENCE
A JOHN WILEY & SONS, INC., PUBLICATION

Copyright © 2008 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at http://www.wiley.com/go/permission.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic format. For information about Wiley products, visit our web site at www.wiley.com.

Library of Congress Cataloging-in-Publication Data is available.

ISBN 978-0-471-27455-1

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

CONTENTS

FOREWORD

The pace of change in software-intensive systems continues to accelerate at a dizzying rate. This presents a huge challenge for people trying to develop useful software. In the early days of software development, developers could freeze the requirements for the software, develop the software to the requirements, and deliver the resulting software two years later with confidence that the requirements would still be relevant and the software would be useful. Most of our software engineering processes, methods, and tools were developed and used under the assumption of relatively stable requirements. Examples are formal specification languages, performance-optimized point-solution designs, fixed-requirements software-cost estimation, earned-value management systems, requirements traceability matrices, fixed-price/fixed-requirements contracts, and a general attitude that "requirements creep" was bad in that it destabilized software development.

However, as these practices became increasingly institutionalized, the accelerating rate of software change made them increasingly risky to use. Projects would use them for two years and become extremely frustrated when the users were not interested in the obsolete capabilities that resulted. Projects would fall behind schedule and use static models (time to complete = work remaining divided by work rate) to try to make up time by adding people, and run afoul of Brooks's law (adding people to a late software project will make it later). Or they would sprint for the finish line using a point-solution design that satisfied the initial requirements but was extremely difficult to modify when trying to satisfy users' changing requirements.

Ironically, even with all of these difficulties, organizations would increasingly turn to software and its ability to be electronically upgraded as their best way to adapt their products, services, and systems to the increasing pace of change in their business or operational environment.

In order to keep up with this increased demand for software and the rapid pace of change, software organizations and projects need better ways to reason about the effects of change on their software products, projects, and processes. This is often very difficult to do, as software changes often have complex second-order and higher-order interaction effects that are hard to visualize and reason about. Thus, for example, a project with high rates of change in its requirements, being developed by a team with high rates of change in its personnel, will need to understand and control the interactions of decreased productivity due to change processing, decreased productivity due to new staff members' unfamiliarity with the project and product, and loss of productivity when key staff members are diverted from project work to train and mentor new people in order to develop increased downstream productivity.

One of the best techniques for reasoning about the effects of such complex interacting changes is the System Dynamics modeling framework that Ray Madachy presents in this book. As I have found in numerous applications of the method, it enables project personnel to model such effects and run the models to better understand the implications of candidate project strategies and decisions.

From the pioneering application of Jay Forrester's System Dynamics approach to software project modeling by Tarek Abdel-Hamid and Stuart Madnick in their 1991 book *Software Project Dynamics,* system dynamics modeling has been applied to many aspects of software development and management. These include the analysis of the effects on software system cost, schedule, and quality of alternative approaches to software process sequencing, software requirements determination, software architecture and design, software project organization and staffing, software development and integration, software quality assurance, and software change management. These applications have constituted the major source of solution approaches in the software process simulation community and its annual series of ProSim conferences (recently combined with the Software Process Workshop into the International Conference on Software Process, held concurrently with the International Conference on Software Engineering).

Ray Madachy has been a major contributor to this body of work. His experience in applying system dynamics modeling as a technical leader in such diverse organizations as the aerospace corporation Litton Systems, the e-commerce system development company C-Bridge Institute, and the software tools company Cost Xpert Group have given him a broad and deep perspective on the critical success factors of developing and applying system dynamics models to various classes of software decision situations. His experience in teaching and researching these techniques at USC has enabled him to develop an integrating framework and set of techniques that make system dynamics modeling much easier and cost-effective to learn and apply to a software decision situation.

His resulting book begins with an overview of the systems dynamics modeling framework, and an example application showing how to develop a system dynamics model that helps explain the conditions under which you can add people to a software project with or without having Brooks's Law apply. Next is an extensive chapter on the modeling process that introduces concepts and techniques used to develop system dynamics models, with illustrations of how they are developed and applied. The next

chapter provides and explains a full range of model structures from modeling molecules to large infrastructures and flow chains that can be used in new models. Once this framework is established, there are three chapters that apply it to the most significant classes of system dynamics applications.

The first of these chapters covers useful applications to software personnel decisions: workforce levels and team composition, learning, burnout, skills, motivation, retention, and rotation effects. The second chapter covers software process and product decision situations: peer reviews, software reuse, COTS-based system development, software architecting, and software process improvement effects. The third chapter covers project and organization applications, such as business case analysis, defect reduction strategies, and staffing management strategies. The final chapter projects likely future uses and research challenges in software process modeling and simulation, including how system dynamics models can add value via integration with other classes of models.

The appendices also provide important and one-of-a-kind material. The first appendix covers statistics of simulation, which is not covered in traditional references on system dynamics. Next is a thorough annotated bibliography of work using system dynamics for software processes, and is the definitive compendium for the field. Finally, the last appendix lists executable models provided with the book. These are used in examples and can be used for further exercises or for incorporation into your own models. These are valuable, reusable, and fun assets to go along with the book.

Overall, the book brings together a tremendous amount of useful process modeling material and experience in using it in practical software decision situations. It organizes this material into a unifying framework that makes it easier to apply and explain, and illustrates it with a wide variety of useful examples. I believe that the book will serve as a standard reference for the software process dynamics field and a great help to practitioners and researchers for a good long time.

BARRY BOEHM
University of Southern California
Los Angeles, California
June 2006

PREFACE

This book is designed for professionals and students in software engineering or information technology who are interested in understanding the dynamics of software development, or in assessing and optimizing process strategies. Its purpose is to improve decision making about projects and organizational policies by making its readers better informed about the dynamic consequences of decisions. Decisions may involve setting project budgets and schedules; return-on-investment analysis; trade-offs between cost, schedule, and quality or other factors; personnel hiring; risk management decisions; make, buy, or reuse; process improvement strategies; and so on.

The importance of process dynamics is hard to refute given the well-known (but too often ignored) combined effects of schedule pressure, communication overhead, changing business conditions, requirements volatility and user requests, experience, work methods such as reviews and quality assurance activities, task underestimation, bureaucratic delays, organizational shifts, demotivating events, other sociotechnical phenomena, and the feedback therein. These complex and interacting process effects are elegantly modeled with system dynamics using continuous quantities interconnected in loops of information feedback and circular causality. Knowledge of the interrelated technical and social factors coupled with simulation tools can provide a means for organizations to improve their processes.

The objectives of this book are to:

- Provide methods, tools, models, and examples to improve management decision making at all levels. Simulation can support corporate strategy and investment analysis, business case development, project and portfolio management, and training, for example.

- Illustrate *systems thinking* in action to develop increasingly deep understandings of software process structures and behaviors.
- Describe the modeling process, including calibration of models to software metrics data.
- Show basic building blocks and model infrastructures for software development processes.
- Review the field of software process modeling with system dynamics. Show how others have used the principles of system dynamics to analyze and improve processes in organizations.
- Provide practical lessons learned about the dynamics of software processes.
- Provide sufficient introductory material, including exercises and executable models on the Internet. Software practitioners who are brand new to simulation can immediately get their hands dirty and start modeling. Students can learn at their own pace, delving into the models as deeply as time and interest dictate.
- For those experienced in software process simulation, provide more detail of critical implementation issues and future research motivations.

The book is mostly new material, except for some example applications, and synthesizes previous work in the area. There has been much growth in the field; it has evolved to a state of maturity, and this book addresses the need to communicate findings. It draws from over 100 publications from practitioners and researchers experienced in system dynamics modeling of processes in organizations (all of them summarized in Appendix B). It is written to be a self-contained learning experience, and a comprehensive reference for modeling practitioners. The sections are structured so that readers can approach the subject from different perspectives and gain valuable knowledge for further study and practice depending on their needs.

A constructive understanding of process dynamics is provided by the illustrated models. Where appropriate, guidelines are presented for process improvement and general software management strategies (common threads include risk management and concentrating on people). The perspective in the book addresses the *dynamics* of software development, and best practices are described from that view. Some of these practices are illuminated through simulation experiments herein, and some will become foci of further study.

Readers may be involved in software process improvement, project planning and management, software development, testing, quality assurance, strategic corporate planning, organizational learning, education, or simply desire to understand the interrelated factors of software development. There is no need for sophisticated math skills, but a passing knowledge of numerical integration concepts will make the introductory material easier. Readers will increase their understanding of the complexities of software development and be able to use system dynamics for modeling and improving processes in their particular organizations. They will gain insight into the real-world mechanics behind the modeling equations.

For academic uses, this book may serve as an upper-level or graduate textbook for Software Process Modeling or other simulation courses. It can be used to support cur-

riculums in Software Engineering, Software Project Management, Software Quality Assurance, Systems Engineering or related subjects.

Part 1 of the book presents modeling fundamentals for software processes. These chapters may constitute an entire course for those new to system dynamics modeling. Advanced students should cover applications and future directions in Part 2. These topics can be studied in whole or as selected subjects. Other disciplines or focused studies may choose relevant application topics. For example, researchers in organizations or sociology might want to cover people applications, engineering process groups might investigate selected process applications, while senior managers could focus on project or organizational applications.

The sequence and depth of subjects should be tailored accordingly for any of these uses. The variety of exercises in the book may serve as homework assignments, exam questions, or even major research projects. Except for the introductory chapters, detailed equations are generally omitted from the text and left for the reader to study from the models.

Though a primary objective is to instruct in computer-aided analysis, several identified exercises early in the book should be done without a computer. This is to help develop intuition of process dynamics, and to strike a balance by not becoming overly reliant on blindly using the computer during model development and evaluation of simulation results. The structure of the book is explained in more detail below.

BOOK ORGANIZATION AND HIGHLIGHTS

This section provides a sequential outline of topics with selected highlights. Each chapter includes numerous graphics, charts, and tables to help illustrate the material. The book is also supplemented on the Internet containing the sample models and simulation tools, exercises, extra references, and updates to the material. The book is divided into two major parts per the outline below:

Part 1—Fundamentals
- Chapter 1—Introduction and Background
- Chapter 2—The Modeling Process with System Dynamics
- Chapter 3—Model Structures and Behaviors for Software Processes

Part 2—Applications and Future Directions
- Introduction to Applications Chapters
- Chapter 4—People Applications
- Chapter 5—Process and Product Applications
- Chapter 6—Project and Organization Applications
- Chapter 7—Current and Future Directions

Appendices and References
- Appendix A—Introduction to Statistics of Simulation
- Appendix B—Annotated Bibliography
- Appendix C—Provided Models
- References

Chapter 1 establishes the context and foundation of the book, with a goal of helping people use models to quantitatively evaluate processes in order to make better decisions. The chapter presents an introduction and background including motivational issues and a capsule history of the field. Definitions of terms are provided for reference throughout the book. The concepts of systems thinking are introduced, so one can see how simulation can be used to leverage learning efforts and improve organizational performance. Control systems principles are introduced, and then a simple motivational example of modeling Brooks's Law is shown. A review of software process technology covers process modeling, lifecycle models, and process improvement.

A description of the iterative modeling process with the system dynamics simulation methodology is provided in Chapter 2. Basic modeling elements and classical system behaviors are shown. The underlying mathematical formulation of system dynamics is covered with its ramifications for software process models.

The activities of problem definition, model formulation (including calibration), simulation, assessment, communication to others, and challenging the model for the next iteration are elaborated on. Since simulation is both art and science, guidelines and modeling heuristics are discussed. It is seen that there is much in common with software engineering principles in general such as iteration, abstraction, aggregation, and so on, yet there are also aspects of simulation that require somewhat different skills.

This chapter also details the multiperspective validation of system dynamics models, which is of paramount importance before drawing policy conclusions from simulation experiments. Different modeling tools and environments are overviewed to help modelers in choosing appropriate tools for their different needs. Also see Appendix A on the use of statistics in the modeling process.

Chapter 3 presents patterns of model structures and behaviors for software processes. Included is a detailed description of levels, flows, auxiliaries, infrastructures, and feedback loops instantiated for software processes. State variables of interest include software work artifacts, defect levels, personnel levels, effort expenditure, schedule date, and others. Corresponding rates over time include software productivity, defect introduction and elimination rates, financial flows for costs and revenue, and so on. Project reference behaviors for different structures and management policies are introduced.

An important contribution of this chapter is the explication of basic flow processes for software development. Common structures for software processes ferreted out of the major models (upcoming in Chapters 4 through 6) are shown. Together with prototypical feedback loops such as learning and project controlling, these infrastructures can be (re)used to develop models relevant to any software process. This section also illustrates a major advantage in system dynamics models over other modeling techniques: inherent cost, schedule, and quality trade-offs by modeling their interactions.

Part 2 covers modeling applications in the field and future directions. Chapter 4 focuses on people applications, Chapter 5 covers process and product applications, and Chapter 6 is about projects and organizations. Each chapter contains applications of varying complexity. An overview of applications and research to date is provided, including history, a list of different implementations, and critiques of the various work.

Modeling examples from the field are shown with sample insights. The examples are further instances of the generic structures from Chapter 3.

The application examples show threads of simulation modeling with actual model implementations and worked out examples. These original examples should be of particular value to system dynamics novices, and more experienced modelers can study them for additional ideas. Many also amplify some lessons learned regarding the software process. Some of the example models are also contained on the accompanying website. Additional exercises are provided for students to work out and practitioners to implement. Note that the applications chapters will also be updated online to keep up with new work.

Chapter 7 presents current and future directions in software process modeling and simulation. These include advances in simulation environments and tools, model structures and component-based model development, new and emerging trends for application models, model integration (not just system dynamics models), empirical research, theory building, and putting it all together in process mission control centers and training facilities.

Appendix A introduces statistics for simulation as an addendum to the modeling fundamentals about which simulation analysts, researchers, and graduate students studying broader aspects of simulation must be knowledgeable. Statistical methods are used to handle the stochastic inputs and outputs of simulation models. The appendix covers the principles of probability distributions, sample size, confidence intervals, and experimental design applied to continuous system simulation. Monte Carlo simulation is described and recommended probability distributions for software process modeling are also provided.

Appendix B is an annotated bibliography of using system dynamics for software processes and is the most complete set of references for the field. It demonstrates well the breadth of applications to date and is a convenient place to start researching particular topics. These same citations are identified in the References in boldface.

Appendix C lists the provided models referenced in the chapters or exercises. These go along with the examples, and can be executed and modified by readers for their own purposes. These models will be updated and replaced on the Internet as improvements are made. Models provided by other readers will also be posted.

INTERNET SITES

The referenced models, tools, updates, discussion, and color book information are available on the world wide web at http://csse.usc.edu/softwareprocessdynamics and at a mirror site http://softwareprocessdynamics.org.

ACKNOWLEDGMENTS

I would like to extend sincere appreciation to all the other people who contributed to this work. I initially learned system dynamics for physiological modeling in a graduate

biomedical engineering course at UCSD in 1982 under the excellent direction of Drs. Alan Schneider and James Bush. This book would not be complete without the accomplishments of other researchers and support of colleagues including Dr. Tarek Abel-Hamid, Richard Adams, Dr. Vic Basili, Dr. James Collofello, Scott Duncan, Dr. Susan Ferreira, Dr. David Ford, Tobias Haberlein, Jim Hart, Dr. Dan Houston, Dr. Marc Kellner, Peter Lakey, Dr. Manny Lehman, Dr. Robert Martin, Dr. Margaret Johnson, Emily oh Navarro, Dr. Nathaniel Osgood, Dr. Dietmar Pfahl, Oliver Pospisil, Dr. David Raffo, Dr. Juan Ramil, Dr. Stan Rifkin, Dr. Howard Rubin, Dr. Ioana Rus, Dr. Walt Scacchi, Dr. Neil Smith, Dr. Greg Twaites, Dr. Wayne Wakeland, Dr. Gerry Weinberg, Dr. Paul Wernick, and Ed Yourdon; Litton personnel including Dr. Denton Tarbet, Wayne Sebera, Larry Bean, Frank Harvey, and Roy Nakahara; Charles Leinbach from C-bridge Institute; Benny Barbe from Cost Xpert Group; and Dr. Julian Richardson and Dr. Michael Lowry for their support at NASA. USC graduate students who contributed to this work are Ashwin Bhatnagar, Cyrus Fakharzadeh, Jo Ann Lane, Dr. Nikunj Mehta, Kam Wing Lo, Jason Ho, Leila Kaghazian, Dr. Jongmoon Baik (also including post-graduate contributions), and Wook Kim. Profound thanks goes to Dr. Barry Boehm, who has served as a mentor and been my biggest influence since the middle of my Ph.D. studies. This book owes much to his continual support, penetrating insights, and inspiration to contribute. Many thanks to the anonymous IEEE reviewers for their long hours and detailed constructive reviews, and IEEE staff including Jeanne Audino, Cathy Faduska, Chrissy Kuhnen, Cheryl Baltes, and Matt Loeb. I also am most grateful to my wife Nancy for her long-term support and lots of patience, and my young daughters Zoey and Deziree for extra motivation and lessons on adaptation to change.

BOOK UPDATES AND MAKING CONTRIBUTIONS

The field of software process modeling itself is quite dynamic, with much happening in conjunction with other software process work. It has been a challenge keeping up with the times as this book has progressed, and the rate of change in the industry has increased over these years. It is inevitable that some things will continue to change, so the reader is urged to access the Internet site for updates at any time, including new and improved models.

Updates to the chapters will be put on the book's Internet site until the next published edition. The application Chapters 4–6 will have substantial updates and entire sections replaced. The goal is to keep the applications current and presented in a uniform format. Chapter 7 on current and future directions is a wild card in terms of predicted changes, and the annotated bibliography will be updated continuously.

It is an exciting time with much opportunity and more work to be done. Hopefully, some of the ideas and exercises in this book will be used as a basis for further practice and research. People will provide new and better exercises and those will be posted too. Your comments on this book and experiences with modeling actual processes are of great interest to this author, and your feedback will help in developing the next edi-

tion. You are encouraged to send any ideas, improvement suggestions, new and enhanced models, or worked out exercises from this book. They will be included in future editions as appropriate.

RAYMOND J. MADACHY

Los Angeles, California
November 2007

Part 1

FUNDAMENTALS

1

INTRODUCTION AND BACKGROUND

Everything is connected to everything.
—Anonymous

Software and information technology professionals, managers, executives, and business analysts have to cope with an increasingly dynamic world. Gone are the days when one's software technology, hardware platforms, organizational environment, and competitive marketplace would stay relatively stable for a few years while developing a system. Thus, the ability to understand and reason about dynamic and complex software development and evolution processes becomes increasingly valuable for decision making.

Particularly valuable are automated aids built upon knowledge of the interacting factors throughout the software life cycle that impact the cost, schedule, and quality. Unfortunately, these effects are rarely accounted for on software projects. Knowledge gleaned from a global perspective that considers these interactions is used in executable simulation models that serve as a common understanding of an organization's processes. Systems thinking, as a way to find and bring to light the structure of the organizational system that influences its dynamic behavior, together with system dynamics as a simulation methodology, provide critical skills to manage complex software development.

System dynamics provides a rich and integrative framework for capturing myriad process phenomena and their relationships. It was developed over 40 years ago by Jay

Software Process Dynamics. By Raymond J. Madachy

Copyright © 2008 the Institute of Electrical and Electronics Engineers, Inc.

Forrester at MIT to improve organizational structures and processes [Forrester 1961]. It was not applied in software engineering until Tarek Abdel-Hamid developed his dissertation model, which is featured in the book *Software Project Dynamics* [Abdel-Hamid, Madnick 1991].

Simulation usage is increasing in many disparate fields due to constantly improving computer capabilities, and because other methods do not work for complex systems. Simulations are computationally intensive, so they are much more cost-effective than in the past. Simulation is general-purpose and can be used when analytic solutions are extremely difficult if not impossible to apply to complex, nonlinear situations. Simulation is even more powerful with improved data collection for the models. Example areas where increased processing power combined with improved models and data include meteorology to better predict hurricane paths, environmental studies, physical cosmology, chemistry to experiment with new molecular structures, or archaeology to understand past and future migrations. These are practical applications but simulation can also be used for experimentation and theory building.

The simulation process in an organization involves designing a system model and carrying out experiments with it. The purpose of these "what if" experiments is to determine how the real or proposed system performs and to predict the effect of changes to the system as time progresses. The modeling results support decision making to improve the system under study, and normally there are unintended side effects of decisions to consider. The improvement cycle continues as organizational processes are continually refined.

Simulation is an efficient communication tool to show how a process works while stimulating creative thinking about how it can be improved. The modeling process itself is beneficial; it is generally acknowledged that much of the reward of modeling is gained in the early stages to gather data, pose questions, brainstorm, understand processes, and so on.

There are many practical benefits of performing simulation in organizations. Besides individual project planning, simulation can help evaluate long-run investment and technology strategies. Companies can use simulation for continuous process improvement, regardless of their current process maturity. It can support organizational learning by making models explicit in a group setting, where all participants can contribute and buy into the model. Such collaboration can go a long way to effect team-building.

Simulation can also be used in individual training, since participants can interact with executing models in real time to see the effects of their decisions. Simulations are used extensively for training in aerospace, military, and other fields. Student awareness is heightened when virtual "games" with simulations are used, particularly when they participate interactively. Visual dynamic graphs or virtual rendering provide faster and more easily remembered learning compared to the traditional lecture format. Exploration is encouraged through the ability to modify and replay the models.

Another significant motivation is that simulation can help reduce the risk of software development. Particularly when used and cross-checked with other complementary analyses that embody different assumptions, process modeling can minimize the

uncertainties of development. Previously unforeseen "gotchas" will be brought to the forefront and mitigated through careful planning.

System dynamics modeling can provide insights by investigating virtually any aspect of the software process at a macro or micro level. It can be used to evaluate and compare different life-cycle processes, defect detection techniques, business cases, interactions between interdisciplinary process activities (e.g. software and nonsoftware tasks), deciding "how much is enough" in terms of rigor or testing, and so on. Organizations can focus on specific aspects of development cost, schedule, product quality, or the myriad trade-offs, depending on their concerns.

The issues of software processes are very wide-ranging, so the scope and boundaries of this book will be defined. The focus is not on technical fundamentals of software programming or specific methodologies, but on the *dynamics* of software processes. The second definition from Webster's dictionary describes the prime focus of this book, particularly the relations between forces:

> *Dynamics*—1. The branch of mechanics dealing with the motions of material bodies under the action of given forces **2. a) the various forces, physical, moral, economic, etc. operating in any field b) the way such forces shift or change in relation to one another c) the study of such forces.**

Essentially, this book is about understanding the dynamics of software processes with the help of simulation modeling. *Software process dynamics* is a more general term than *software project dynamics,* which is limiting in the sense that dynamics occur outside of project boundaries such as continuous product line development, organizational reuse processes contributing to many projects, or other strategic processes. A project is also considered an execution of a process, roughly analogous to how a programming object is an instance or execution of a class.

When simulation is used for personnel training, the term *process flight simulation* is sometimes used to invoke the analogy of pilots honing their skills in simulators to reduce risk, with the implicit lesson that software managers and other personnel should do the same. Use of the system dynamics method may on occasion be referred to as *dynamic process simulation, dynamic simulation,* or *continuous systems simulation.*

Alternative titles for this book could be *The Learning Software Organization* or *Software Process Systems Thinking,* depending on the camp de jour. System dynamics and, particularly, organizational learning gained wider public exposure due to Peter Senge's bestselling book *The Fifth Discipline* [Senge 1990]. Organizational learning in the context of a software process involves translating the common "mental model" of the process into a working simulation model that serves as a springboard for increased learning and improvement. This learning can be brought about by applying system dynamics to software process and project phenomena.

There are other excellent references on system dynamics modeling that one could use to learn from, but why should a busy software engineer studying the software process spend so much time with examples outside of his/her field? This book uses examples solely from the software process domain to minimize modeling skill transfer time. Organizational learning and systems thinking are also well documented else-

where (see the popular books by Peter Senge and collaborators [Senge 1990], [Senge et al. 1994]).

1.1 SYSTEMS, PROCESSES, MODELS, AND SIMULATION

Important terminology for the field is defined in this section. A systems orientation is crucial to understanding the concepts herein, so system will first be defined generally as a subset of reality that is a focus of analysis. Technically, systems contain multiple components that interact with each other and perform some function together that cannot be done by individual components. In simulation literature, a system is typically defined as "a collection of entities, e.g., people or machines, that act and interact together toward the accomplishment of some logical end" [Law, Kelton 1991]. Forrester's system definition is very close: "a grouping of parts that operate together for a common purpose" [Forrester 1968].

Systems exist on many levels; one person's system is another person's subsystem. Since systems are influenced by other systems, no system is isolated from external factors. How to define system boundaries for meaningful analysis is discussed later in this book.

Systems are classified as "open" if the outputs have no influence on the inputs; open systems are not aware of their past performance. A "closed" system is also called a feedback system; it is influenced by its own behavior through a loop that uses past actions to control future action. The distinction between open and closed systems is particularly important in the context of system dynamics.

A system can be characterized by (1) parameters that are independent measures that configure system inputs and structure, and (2) variables that depend on parameters and other variables. Parameters in human systems are directly controllable. The collection of variables necessary to describe a system at any point in time is called the *state* of the system. Examples of state variables for a software process are the number of personnel executing the process; the amount of software designed, coded, and tested; the current number of defects; and so on.

Real-world systems can be classified as static or dynamic depending on whether the state variables change over time. The state of a static system does not change over time, whereas the state of a dynamic system does. Dynamic systems can be further classified as continuous, discrete, or combined, based on how their variables change over time.

Variables change continuously (without breaks or irregularities) over time in a continuous system, whereas they change instantaneously at separated time points in a discrete system. A lake is an example of a continuous system since its depth changes continuously as a function of inflows and outflows, whereas a computer store queue would be considered discrete since the number of customers changes in discrete quantities. A software process arguably has continuous quantities (personnel experience, motivation, etc.) and discrete ones (lines of code, defects, etc.)

Whether a system is seen as continuous, discrete, or combined depends on one's perspective. Furthermore, the choice of a continuous or discrete representation de-

pends on the modeling purpose, and some discrete systems can be assumed to be continuous for easy representation. For example, some would consider a software process to be a system with discrete entities since it can be described by the number of people working, number of units/lines/objects produced, defects originated, and so on, but much difficulty will be avoided if each entity does not need to be traced individually. Hence, the approach in this book and system dynamics in general is to treat the "flow" of the software process as continuous.

A software process is a set of activities, methods, practices, and transformations used by people to develop software. This is a general definition from the commonly accepted Software Engineering Institute's Capability Maturity Model (SEI CMM) [Paulk et al. 1994]. In the context of this book, the software process is the system under study.

A system must be represented in some form in order to analyze it and communicate about it. A *model* in the broadest sense is a representation of reality, ranging from physical mockups to graphical descriptions to abstract symbolic models. Software programs are themselves executable models of human knowledge. A model in the context of this book is a logical, quantitative description of how a process (system) behaves. The models are abstractions of real or conceptual systems used as surrogates for low cost experimentation and study. Models allow us to understand a process by dividing it into parts and looking at how they are related.

Dynamic process models can be discrete, continuous, or a combination of the two. The essential difference is how the simulation time is advanced. Continuous systems modeling methods such as system dynamics always advance time with a constant delta. Since variables may change within any time interval in a continuous system, the delta increment is very small and time-dependent variables are recomputed at the end of each time increment. The variables change continuously with respect to time. Discrete modeling normally is event based. State changes occur in discrete systems at aperiodic times depending on the event nature, at the beginning and end of event activities. The simulation time is advanced from one event to the next in a discrete manner.

All classes of systems may be represented by any of the model types. A discrete model is not always used to represent a discrete system and vice versa. The choice of model depends on the specific objectives of a study. Models of the software processes are either static,[1] in which time plays no role, or dynamic, in which a system evolves over time. The dynamic process models described this book are classified as symbolic, or mathematical ones.

Models may be deterministic, with no probabilistic components, or stochastic, with randomness in the components. Few, if any, software processes are wholly deterministic. Stochastic models produce output that is random and must be handled as such with independent runs. Each output constitutes an estimate of the system characteristics.

[1]A cost model such as COCOMO II [Boehm et al. 2000] is traditionally a static model since the cost factors are treated as constant for the project duration. However, there is a continuum between static and dynamic versions of COCOMO. There are variations that make it possible to introduce time into the calculations.

Simulation is the numerical evaluation of a mathematical model describing a system of interest. Many systems are too complex for closed-form analytical solutions, hence, simulation is used to exercise models with given inputs to see how the system performs. Simulation can be used to explain system behavior, improve existing systems, or to design new systems too complex to be analyzed by spreadsheets or flowcharts.

Finally, *system dynamics* is a simulation methodology for modeling continuous systems. Quantities are expressed as levels, rates, and information links representing feedback loops. Levels represent real-world accumulations and serve as the state variables describing a system at any point in time (e.g., the amount of software developed, number of defects, number of personnel on the team, etc.) Rates are the flows over time that affect the levels. See Table 1.3-1 for a preview description of model elements. System dynamics is described in much more detail in Chapter 2.

A complete and rigorous reference for terms related to modeling and simulation can be found at [DMSO 2006].

1.2 SYSTEMS THINKING

Systems thinking is a way to ferret out system structure and make inferences about the system, and is often described as an overall paradigm that uses system dynamics principles to realize system structure. Systems thinking is well suited to address software process improvement in the midst of complexity. Many organizations and their models gloss over process interactions and feedback effects, but these must be recognized to effect greater improvements.

Systems thinking involves several interrelated concepts:

- A mindset of thinking in circles and considering interdependencies. One realizes that cause and effect can run both ways. Straight-line thinking is replaced by closed-loop causality.
- Seeing the system as a cause rather than effect (internal vs. external orientation). Behavior originates within a system rather than being driven externally, so the system itself bears responsibility. It is the structure of a system that determines its dynamic behavior.
- Thinking dynamically in terms of ongoing relationships rather than statically.
- Having an operational vs. a correlational orientation; looking at how effects happen. Statistical correlation can often be misleading. A high correlation coefficient between two factors does not prove that one variable has an impact on the other.

Systems thinking is, therefore, a conceptual framework with a body of knowledge and tools to identify wide-perspective interactions, feedback, and recurring structures. Instead of focusing on open-loop, event-level explanations and assuming cause and effect are closely related in space and time, it recognizes the world really consists of multiple closed-loop feedbacks, delays, and nonlinear effects.

1.2.1 The Fifth Discipline and Common Models

Senge discusses five disciplines essential for organizational learning in [Senge 1990]: personal mastery, mental models, shared vision, team learning, and systems thinking. Systems thinking is the "fifth" discipline that integrates all the other disciplines and makes organizational learning work. Improvement through organizational learning takes place via shared mental models.

Mental models are used in everyday life for translating personal or organizational goals into issues, questions, and measures. They provide context for interpreting and acting on data, but seldom are stated explicitly. Mental models become more concrete and evolve as they are made progressively explicit. The power of models increases dramatically as they become more explicit and commonly understood by people; hence, process modeling is ideally suited for organizational improvement.

For organizational processes, mental models must be made explicit to frame concerns and share knowledge among other people on a team. Everyone then has the same picture of the process and its issues. Senge and Roberts provide examples of team techniques to elicit and formulate explicit representations of mental models in [Senge et al. 1994]. Collective knowledge is put into the models as the team learns. Elaborated representations in the form of simulation models become the bases for process improvement.

1.2.2 Systems Thinking Compared to System Dynamics

Systems thinking has been an overloaded term in the last 15 years with many definitions. Virtually any comparison with system dynamics is bound to be controversial due to semantic and philosophical issues. Barry Richmond addressed the differences between systems thinking and system dynamics mindsets in detail in [Richmond 1994a]. His major critique about "the historical emphasis of system dynamics" is that the focus has been on product rather than transferring the process (of model building). Only a privileged few developed models and presented them to the world as "the way" as opposed to educating others to model and letting them go at it.[2] His prescription is a systems thinking philosophy of providing skills rather than models per se. Relevant aphorisms include "Give a fish, eat for a day; teach to fish, eat for a lifetime," or "power to the people."

His definition of systems thinking is "the art and science of making reliable inferences about behavior by developing an increasingly deep understanding of underlying structure." It is both a paradigm and a learning method. The paradigm is a vantage point supplemented with thinking skills and the learning method is a process, language, and technology. The paradigm and learning method form a synergistic whole. System dynamics inherently fits in as the way to understand system structure. Thus, system dynamics is a methodology to implement systems thinking and leverage learning efforts.

We prefer not to make any hard distinctions between camps because it is a semantic issue. However, this book is architected in the spirit of systems thinking from the perspective of transferring the process. The goal is to teach people how to model and give

[2]It should be noted encouragingly that the system dynamics pioneer Jay Forrester and others at MIT are involved in teaching how to model with system dynamics in K–12 grades.

them tools to use for themselves, rather than say "here is the model for you to use." This is a major difference between *Software Project Dynamics* and this book. Abdel-Hamid and Madnick present a specific model with no guidance on how to develop a system dynamics model, though very few organizations are content to use the model as-is. Their work is still a seminal contribution and it helped make this book possible.

1.2.3 Weinberg's Systems Thinking

Gerry Weinberg writes about systems thinking applied to software engineering in *Quality Software Management, Volume 1: Systems Thinking* [Weinberg 1992]. It is an insightful book dealing with feedback control and has a close kinship with this book, even though it is almost exclusively qualitative and heuristic. Some academic courses may choose his book as a companion to this one. It provides valuable management insights and important feedback situations to be modeled in more detail.

Weinberg's main ideas focus around management thinking correctly about developing complex software systems—having the right "system model" for the project and its personnel. In a restatement of Brooks's dictum that lack of schedule time has doomed more projects than anything else, Weinberg writes in [Weinberg 1992], "Most software projects have gone awry from management's taking action based on *incorrect system models* than for all other causes combined."

One reason management action contributes to a runaway condition is the tendency to respond too late to deviations, which then forces management to take big actions, which themselves have nonlinear consequences. In order to stay in control of the software process, Weinberg advises to "act early, act small." Managers need to continually plan, observe the results, and then act to bring the actuals closer to planned. This is the prototypical feedback loop for management.

Weinberg was working on his book at the same time that Abdel-Hamid and Madnick were working on theirs, unknown to each other. The day after Weinberg submitted his work to the publisher, he met Abdel-Hamid and realized they were working on parallel and complementary paths for years. Weinberg describes the relationship between the two perspectives as follows. He starts from the low end, so projects get stable enough so that the more precise, high-end modeling exemplified by system dynamics can be even more useful.

Much of Weinberg's book discusses quality, on-the-job pressures, culture, feedback effects, dynamics of size and fault resolution, and more. He proceeds to describe the low-level interactions of software engineering, which are the underlying mechanics for many of the dynamic effects addressed by various process models described in this book. His work is referenced later and provides fodder for some exercises.

1.3 BASIC FEEDBACK SYSTEMS CONCEPTS APPLIED TO THE SOFTWARE PROCESS

Continuous systems modeling has a strong cybernetic thread. The word cybernetic derives from "to control or steer," and cybernetics is the field of science concerned with

processes of communication and control (especially the comparison of these processes in biological and artificial systems) [Weiner 1961]. Cybernetic principles are relevant to many types of systems including moving vehicles (ground, air, water, or space), biological systems, individuals, groups of individuals, and species.

We are all familiar with internal real-time control processes, such as when driving down a road. We constantly monitor our car's position with respect to the lane and make small adjustments as the road curves or obstacles arise. The process of monitoring actual position against desired position and making steering adjustments is similar to tracking and controlling a software project. The same mathematics apply, so system dynamics can be used to model the control aspects of either human driving or project management.

Control systems theory provides a rigorous framework for analyzing complex feedback systems. This section will introduce some basic system notations and concepts, and apply to them to our system of study—the software process. The purpose is to realize a high-level analogy of control principles to our domain, and we will forego mathematical formulae and more sophisticated feedback notations.[3] System dynamics is our chosen method for modeling feedback systems in a continuous-time fashion, as used in the rest of this book.

Figure 1.1 shows the most basic representation of an open system, whereby a black-box system transforms input to output per its internal processing functions. Input and output signals are treated as fluxes over time. It is open because the outputs have no system influence (frequently, it is also called an *open-loop* system despite the absence of any explicit loops). Figure 1.2 shows the closed-loop version with a controller implementing feedback. A decomposition of the controller shows two major elements: a sensor and a control device, shown in Figure 1.3.

The borrowing of these standard depictions from control systems theory can lead to misinterpretation about the "system" of interest for software processes. In both Figures 1.2 and 1.3, the controller is also of major concern; it should not be thought of as being "outside" the system. One reason for problems in software process improvement is that management is often considered outside the system to be improved. Therefore, the boundary for a software process system including management should encompass all the elements shown, including the controller.

Applying these elements to the software process, inputs traditionally represent requirement specifications (or capabilities or change requests), the system is the software development (and evolution) process with the management controller function, and the outputs are the software product artifacts (including defects). The sensor could be any means of measuring the output (e.g., analyzing software metrics), and the control device is the management action used to align actual process results with intended. This notation can represent either a one-time project or a continual software evolution process.

If we consider all types of inputs to the software process, the vector includes resources and process standards as well as requirements. Resources include people and

[3]We are not covering signal polarities, integrators, summers, transfer functions, Laplace transforms, cascaded systems, state space representation, and so on.

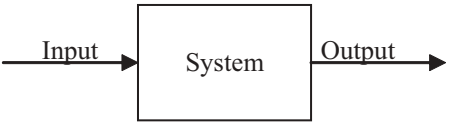

Figure 1.1. Open system.

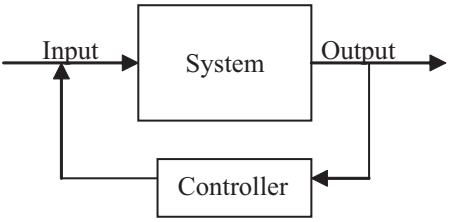

Figure 1.2. Closed system with controller.

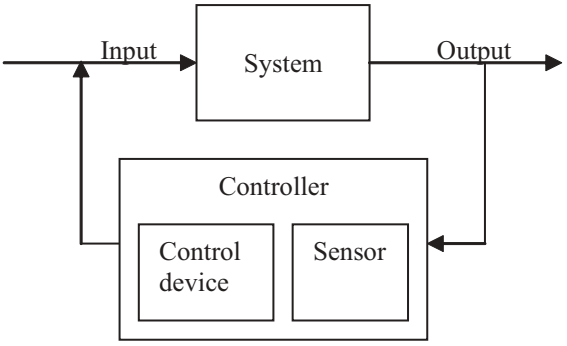

Figure 1.3. Closed system with controller elements.

machines used to develop and evolve the software. Process standards include methods, policies, procedures, and so on. Even the process life-cycle model used for the project can be included (see Section 1.3.2). Requirements include functional requirements, project constraints like budget and schedule, support environment requirements, evolution requirements, and more. Process control actions that management takes based on measurements may affect any of these inputs.

Substituting software process elements into the generic system description produces Figures 1.4, keeping the controller aggregated at the top level representing internal process management.

However, the management controller only represents endogenous process mechanisms local to the development team. These are self-initiated control mechanisms. In reality, there are external, or exogenous feedback forces from the operational environ-

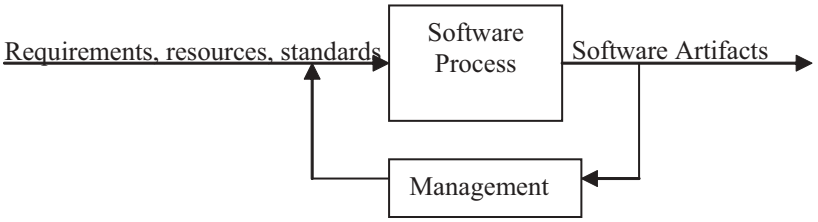

Figure 1.4. Software process control system with management controller.

ment of the software—global feedback. The feedback can be user change requests from the field, other stakeholder change mandates, market forces, or virtually any external source of requirements evolution or volatility. The exogenous feedback is a very important effect to understand and try to control. An enhanced picture showing the two sources of feedback is in Figure 1.5.

The outer, global feedback loop is an entire area of study in itself. Of particular note is the work of Manny Lehman and colleagues on software evolution, which is highlighted in Chapter 5 and referenced in several other places (also see their entries in Appendix B).

These feedback mechanisms shown with control systems notation are implemented in various ways in the system dynamics models shown later. Feedback is represented as information connections to flow rates (representing policies) or other parameters that effect changes in the systems through connected flow rates.

1.3.1 Using Simulation Models for Project Feedback

Projects can proactively use simulation models to adapt to change, thereby taking advantage of feedback to improve through models. This is one way to implement operational control through simulation. A simulation model can be used for metrics-based feedback during project execution since its input parameters represent project objec-

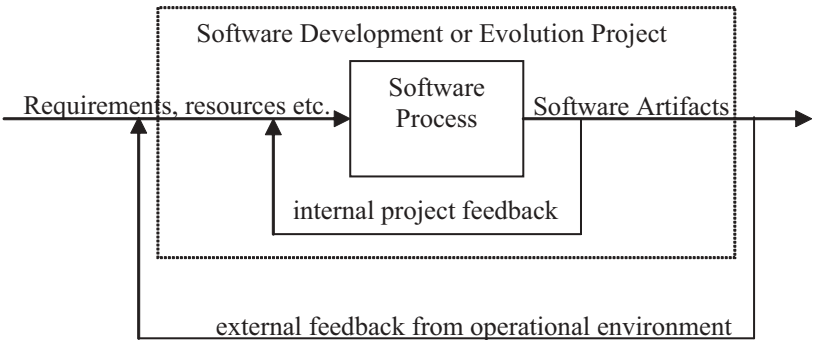

Figure 1.5. Software process control system with internal and external feedback.

tives, priorities, available components, or personnel. It serves as a framework for project rescoping and line management to reassess risks continuously and support replanning.

Figure 1.6 shows a project rescoping framework utilizing metrics feedback and simulation. By inputting parameters representing changed conditions, one can assess whether the currently estimated cost and schedule are satisfactory and if action should be taken. Either rescoping takes places or the project executes to another feedback milestone, where the model is updated with actuals to date and the cycle repeats.

1.3.2 System Dynamics Introductory Example

Table 1.1 is a heads-up preview of system dynamics model elements used throughout this book. The capsule summary may help to interpret the following two examples before more details are provided in Chapter 2 (this table is a shortened version of one in Chapter 2). We are jumping ahead a bit in order to introduce a simple Brooks's Law model. Novices may also want to consult the system dynamics introduction in Chapter 2 to better understand the model elements.

Throughout this text and in other references, levels are synonymous with "stocks" and rates are also called "flows." Thus, a stock and flow representation means an elaborated model consisting of levels and rates.

A simple demonstration example of modeling process feedback is shown in the Figure 1.7 system diagram. In this model, the software production rate depends on the number of personnel, and the number of people working on the project is controlled via a feedback loop. The linear *software production rate* is expressed as

$$\text{software production rate} = \text{individual productivity} \cdot \text{personnel}$$

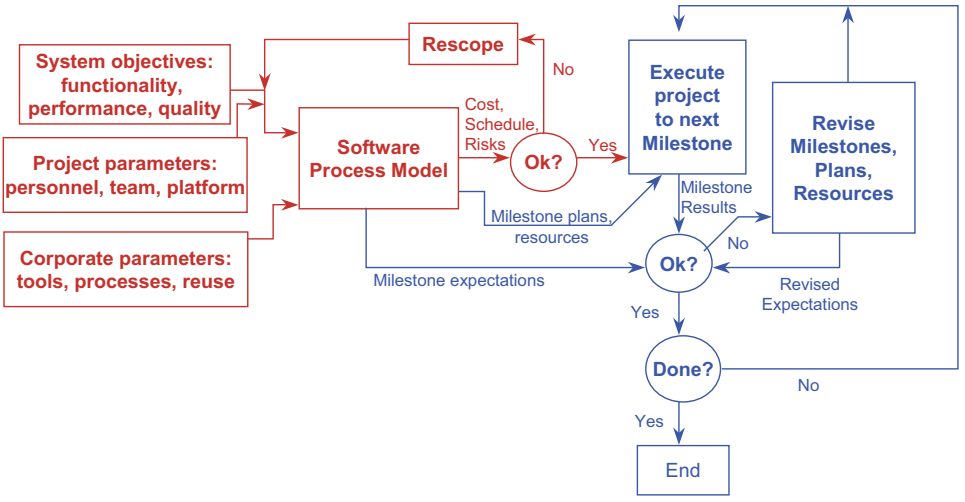

Figure 1.6. Project rescoping framework.