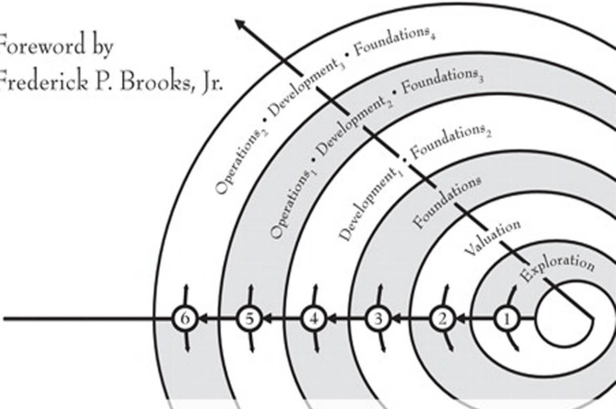
Barry Boehm • Jo Ann Lane
Supannika Koolmanojwong • Richard Turner

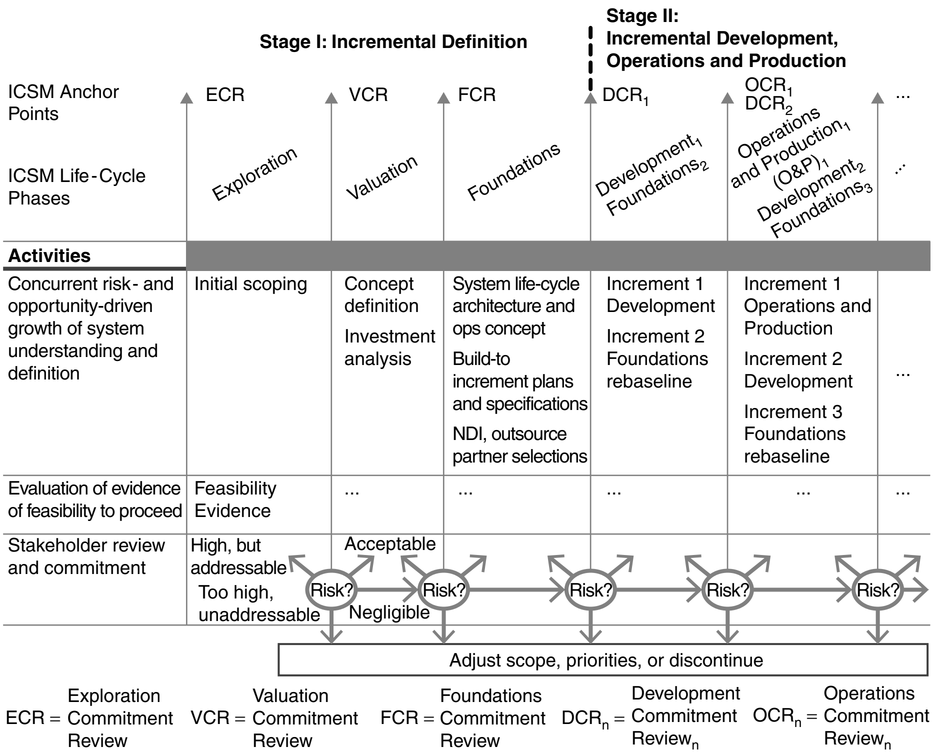
The Incremental Commitment SPIRAL MODEL

Foreword by
Frederick P. Brooks, Jr.



Principles and Practices for Successful Systems and Software

The Incremental Commitment Spiral Model: Phased View



Feasibility Evidence Description Content

Evidence *provided by the developer* and *validated by independent experts* that, if the system is built to the specified architecture, it will:

- Satisfy the requirements: capability, interface, level of service, and evolution
- Support the operational concept
- Be buildable within the budgets and schedules in the plan
- Generate a viable return on investment
- Generate satisfactory outcomes for all of the success-critical stakeholders
- Resolve all major risks by treating shortfalls in evidence as risks and covering them by risk management plans
- Serve as a basis for stakeholders' commitment to proceed

Principles Trump Diagrams

The Four ICSM Principles

1. **Stakeholder value-based guidance.**
2. **Incremental commitment and accountability.**
3. **Concurrent multi-discipline engineering**
4. **Evidence and risk-based decisions.**

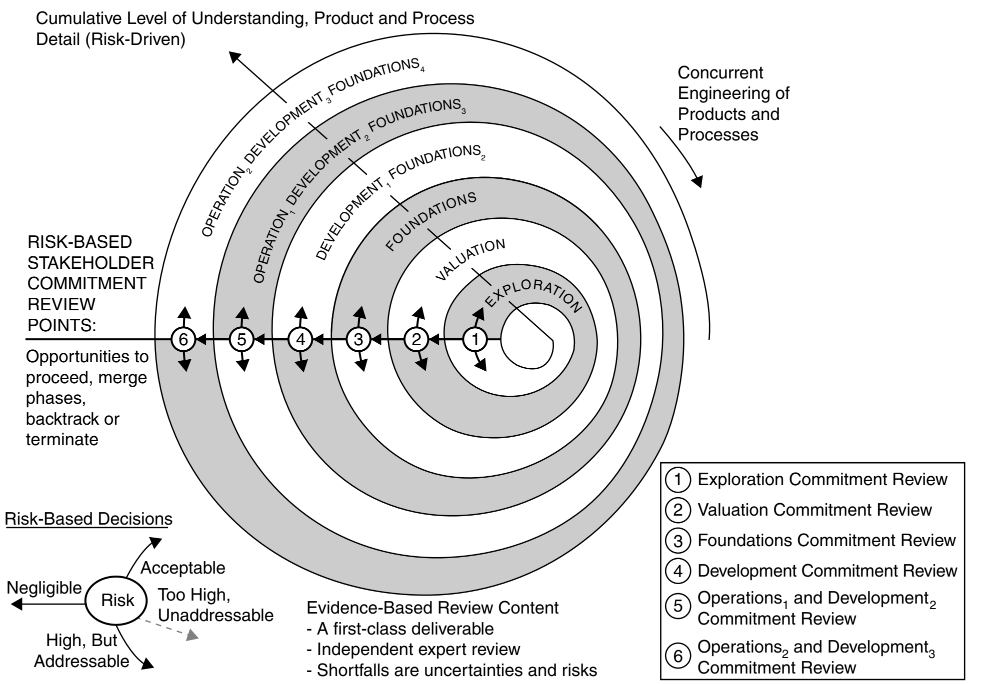
Risk Meta-Principle of Balance: Balancing the risk of doing too little and the risk of doing too much will generally find a middle course sweet spot that is about the best you can do.

Theory W (Win-Win) Success Theorem: *A system will succeed if and only if it makes winners of its success-critical stakeholders.*

System Success Realization Theorem: *Making winners of your success-critical stakeholders requires:*

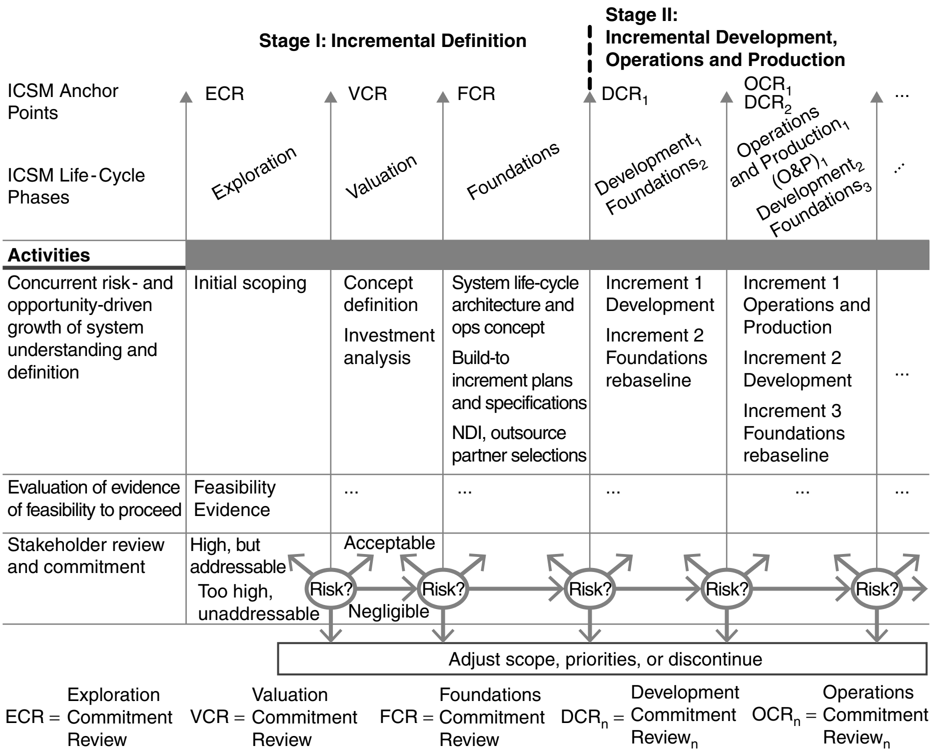
1. Identifying all of the success-critical stakeholders.
2. Understanding how each stakeholder wants to win.
3. Having the success-critical stakeholders negotiate among themselves a win-win set of product and process plans.
4. Controlling progress toward the negotiated win-win realization, including adapting it to change.

The Incremental Commitment Spiral Model



Reprinted with permission from *Human-System Integration in the System Development Process*, 2007 by the National Academy of Sciences, Courtesy of the National Academies Press, Washington, D.C.

Principles Trump Diagrams



Feasibility Evidence Description Content

Evidence *provided by the developer* and *validated by independent experts* that, if the system is built to the specified architecture, it will:

- Satisfy the requirements: capability, interface, level of service, and evolution
- Support the operational concept
- Be buildable within the budgets and schedules in the plan
- Generate a viable return on investment
- Generate satisfactory outcomes for all of the success-critical stakeholders
- Resolve all major risks by treating shortfalls in evidence as risks and covering them by risk management plans
- Serve as a basis for stakeholders' commitment to proceed

This page intentionally left blank

Praise for *The Incremental Commitment Spiral Model*

"*The Incremental Commitment Spiral Model* is an extraordinary work. Boehm and his colleagues have succeeded in creating a readable, practical, and eminently usable resource for the practicing systems engineer. . . . ICSM embodies systems thinking and engineering principles and best practices using real-life examples from many different application domains. This is exactly the kind of treatment that an engineer needs to translate the book's considerable wisdom into practical on-the-job solutions."

—George Rebovich, Jr., Director, Systems Engineering Practice Office, The MITRE Corporation

"One might think of this new book as an update of the old (1988) Spiral Model, but it is actually much more than that. It is a ground-breaking treatment that expertly blends together four specific and key principles, risk–opportunity management, the use of existing assets and processes, and lessons learned from both success and failure examples and case studies. This extraordinary treatise will very likely lead to improvements in many of the current software development approaches and achieve the authors' intent 'to better integrate the hardware, software, and human factors aspects of such systems, to provide value to the users as quickly as possible, and to handle the increasingly rapid pace of change.' If one is looking for specific ways to move ahead, use this book and its well-articulated advancements in the state-of-the-art."

—Dr. Howard Eisner, Professor Emeritus and Distinguished Research Professor, George Washington University

"Dr. Boehm and his coauthors have integrated a wealth of field experience in many domains and created a new kind of life cycle, one that you have to construct based on the constraints and objectives of the project. It is based on actively trading off risks and demonstrating progress by showing actual products, not paper substitutes. And the model applies to everything we build, not just software and conceptual systems, but also to hardware, buildings, and garden plots. We have long needed this experience-based critical thinking, this summative and original work, that will help us avoid chronic systems development problems (late, over-budget, doesn't work) and instead build new life cycles matched to the circumstances of the real world."

—Stan Rifkin, Principal, Master Systems

"Barry Boehm and his colleagues have created a practical methodology built upon the one fundamental truth that runs through all competitive strategies: The organization with the clearest view of cold, brutal reality wins. Uniquely, their methodology at every stage incorporates the coldest reality of them all—the customer's willingness to continue paying, given where the project is today and where it is likely ever to be."

—Chet Richards, author of *Certain to Win: The Strategy of John Boyd Applied to Business*

"I really like the concept of the ICSM and have been using some of the principles in my work over the past few years. This book has the potential to be a winner!"

—Hillary Sillito, INCOSE Fellow, Visiting Professor University of Bristol, formerly Thales UK Director of Systems Engineering

"*The Incremental Commitment Spiral Model* deftly combines aspects of the formerly isolated major systems approaches of systems engineering, lean, and agile. It also addresses perhaps the widest span of system sizes and time scales yet. Two kinds of systems enterprises especially need this capability: those at the 'heavy' end where lean and agile have had little impact to date, and those that deal with a wide span of system scales. Both will find in the ICSM's combination of systems approaches a productive and quality advantage that using any one approach in isolation cannot touch."

—James Maxwell Sutton, President, Lean Systems Society and Shingo Prize winner

"The potential impact of this book cannot be overstressed. Software-intensive systems that are not adequately engineered and managed do not adequately evolve over the systems life cycle. The beauty of this book is that it describes an incremental capability decision path for being successful in developing and acquiring complex systems that are effective, resilient, and affordable with respect to meeting stakeholders' needs. I highly recommend this book as a 'must read' for people directly involved in the development, acquisition, and management of software-intensive systems."

—Dr. Kenneth E. Nidiffer, Director of Strategic Plans for Government Programs, Software Engineering Institute, Carnegie Mellon University

"This text provides a significant advance in the continuing work of the authors to evolve the spiral model by integrating it with the incremental definition and the incremental development and evolution life-cycle stages. Case studies illustrate how application of the four principles and the Fundamental Systems Success Theorem provides a framework that advances previous work. Emphasis is placed throughout on risk-based analysis and decision making. The text concludes with guidance for applying ICSM in your organization plus some helpful appendices. We concur with the authors' statement: 'we are confident that this incarnation of the spiral model will be useful for a long time to come.'"

—Dick Fairley, PhD, Software and Systems Engineering Associates (S2EA)

"This book nicely integrates the different refinements of the spiral model and the various additions made over the years. . . . the book contains great material for classes on software engineering in general and software processes in particular. I have been teaching the spiral model and its invariants for more than 10 years now, and I will use material from this book in the years to come."

—Paul Grünbacher, Associate Professor, Johannes Kepler University Linz, Head of the Christian Doppler Lab for Monitoring and Evolution of Very-Large-Scale Software Systems

"What I found most useful in *The Incremental Commitment Spiral Model* were the stories of where we have gone wrong in the past, and how using the four key ICSM principles articulated by Barry and his co-authors could have helped these failed efforts maintain a course to success. ICSM is not a new method. It does not ask you to discard what has proved useful in the past and start over. Rather, it provides a set of guideposts that can help any organization facing increasingly challenging endeavors make more timely evidence-based decisions. We have been hearing about the 'what' for many years, this book gives you the needed 'how' and, more importantly, the needed 'how much' guidance that has been sorely missing."

—Paul E. McMahon, author of *Integrating CMMI and Agile Development*

"The authors are uniquely qualified to bring together a historical context and a modern problem: successful development of engineered systems with ever greater complexity and richer than ever functionality, enabled by software. They do not disappoint!"

—Dinesh Verma, PhD, Professor and Dean, School of Systems and Enterprises, Stevens Institute of Technology

This page intentionally left blank

The Incremental Commitment Spiral Model

This page intentionally left blank

The Incremental Commitment Spiral Model

Principles and Practices for Successful Systems and Software

Barry Boehm
Jo Ann Lane
Supannika Koolmanojwong
Richard Turner

Addison-Wesley

Upper Saddle River, NJ ▪ Boston ▪ Indianapolis ▪ San Francisco
New York ▪ Toronto ▪ Montreal ▪ London ▪ Munich ▪ Paris ▪ Madrid
Capetown ▪ Sydney ▪ Tokyo ▪ Singapore ▪ Mexico City

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Boehm, Barry W.
The incremental commitment spiral model : principles and practices for successful systems and software / Barry Boehm, Jo Ann Lane, Supannika Koolmanojwong, Richard Turner.
pages cm
Includes bibliographical references and index.
ISBN-13: 978-0-321-80822-6 (pbk. : alk. paper)
ISBN-10: 0-321-80822-3 (pbk. : alk. paper)
1. Computer software—Development. 2. Continuous improvement process. I. Koolmanojwong, Supannika II. Lane, Jo Ann. III. Turner, Richard, 1954 August 18-. IV. Title.
QA76.76.D47B635 2014
005.1—dc23 2014006606

ISBN-13: 978-0-321-80822-6
ISBN-10: 0-321-80822-3
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, May 2014

Contents

Foreword

Developers, thinkers, and writers have wrestled since the 1960s with process models for building software, including my own 1975 simple-minded "Plan to throw one away; you will anyhow." Practitioners in the software development discipline early learned that a patterned development is more likely to succeed than a chaotic one, at any size. Hence, the emergence of process models.

I am firmly convinced that the model set forth in this book is by far the best anyone has developed. First proposed by Boehm in 1988, it was even then the fruit of much thought and a rich trove of practical experience. In the almost 30 years since its introduction, the Incremental Commitment Spiral Model has grown and evolved through actual use in many projects, and through systematic thought. It has been extended from software to systems, and to the larger life cycle.

The most important augmentation of the original spiral model has been the addition of formal, cold-eyed assessments of risk at the various checkpoints. A second important addition is the explicit prescription that the stakeholders regularly and boldly consider abandoning the project. To paraphrase this dictate: "**Plan** to consider throwing the project away; you may need to consider that anyhow." The Preface lists other ways the model has grown.

The work presented in this book demands and repays careful study. The Introduction sets forth the basic concepts of the model and the experienced-based motivations for each refinement. Since what is treated is not itself a model but a model generator, it can be flexibly adapted for projects large and small, long and short. Such adaptation requires thinking, of course.

The organization of the book into individual, self-contained parts suggests the mode of study. Students with no project experience can manage the Introduction and profit from it. The more sophisticated later parts will come to life for those practitioners who have experienced both successful and unsuccessful projects, and who want to ensure that their subsequent ventures are successful ones. They may want to ponder each part as a chunk, fleshing out and coloring the ideas and recommendations with their own experiences.

—Frederick P. Brooks, Jr.
author, *The Design of Design*

Preface

This book describes a way to be successful in an increasingly challenging endeavor: developing systems that are effective, resilient, and affordable with respect to meeting stakeholders' needs. Most people would prefer to be part of creating a successful system. Rumor has it, however, that some people would rather deliver an unsuccessful system so that they can continue being paid to make it successful; rumor also doubts those people will read this book.

We have been studying and experimenting with approaches for creating successful systems for many years and have seen constant evolution in system capability, content, and context. The systems we worked on were initially hardware items such as radios, power supplies, airplanes, and rockets. As time went on, the systems became more software intensive. For example, in some classes of airplanes, the functionality performed by software grew from 8% in 1960 to 80% in 2000. Both now and for the foreseeable future, most systems must interact with other independently evolving systems to help provide additional functionality and flexibility. Even more important, precisely because it has often been overlooked, is the increasing role that humans are playing as system elements, as the enterprise is viewed as a holistic interdisciplinary entity. Perhaps the farthest-reaching change is that so many traditional stand-alone hardware devices need to cope not only with software, but also with living in an Internet of Things, preserving cybersecurity, and adjudicating among human users and smart autonomous agents.

The Incremental Commitment Spiral Model (ICSM) is the result of our efforts to better integrate the hardware, software, and human factors aspects of such systems; to provide value to the users as quickly as possible; and to handle the increasingly rapid pace of change. While the ICSM's pedigree lies in Barry's spiral concept first articulated in 1988, this new version draws on more than 20 years of experience helping people deal with the fact that the original version was too easy to misinterpret. The ICSM is both more general and more specific than the original spiral. It covers more of the life cycle, addresses not only software projects but also cyber–physical–human systems and enterprises, and is adaptable to most development endeavors. At the same time, it is much more specific about how to implement the principles and activities.

The ICSM is not a single, one-size-fits-all process. It is actually a process generator that steers your process in different directions, depending on your particular circumstances. In this way, it can help you adapt your life-cycle strategies and processes to your sources of change. It also supports more rapid system development and evolution through concurrent engineering, enabling you to develop and evolve systems more rapidly and to avoid obsolescence.

If things aren't changing much in your domain, and you already have a way to create successful systems, you should keep on using it. But you will be in a shrinking minority as the 21st-century pace of change accelerates. When you find that your processes are out of step with your needs, we believe you will find the ICSM helpful.

Who Can Benefit from Reading This Book?

The book's contents can help you if you face one or more of the following situations:

- Your projects frequently overrun their budgets and schedule.
- Your projects have a lot of late rework or technical debt.
- Your delivered systems are hard to maintain.
- Your organization uses a one-size-fits all process for a variety of systems.
- Your systems need to succeed in situations involving rapid change, emergent requirements, high levels of assurance, or some combination of those.
- Your systems must operate with other complex, networked systems.

Managers and executives stuck in one-size-fits-all decision sequences will find new possibilities and begin to understand their new roles in successful 21st-century development. Practitioners of all development-related disciplines will find a unified way to approach a broad variety of projects, improve their collaboration, respond more agilely to the changing needs of stakeholders, and better quantify and demonstrate progress to managers and executives. Academics will gain a source of information to replace or enhance the way they educate developers and managers, as well as fertile areas for research and study.

As one-step, total-makeover corporate process changes can be risky, this book provides a way for organizations or projects to incrementally experiment with the ICSM's key practices and to evolve toward process models better suited to their needs and competitive environment.

An Electronic Process Guide (EPG), available on the book's companion website (http://csse.usc.edu/ICSM), contains guidelines, subprocesses, and templates that facilitate ICSM adoption. The EPG also supports this volume's use as a textbook for a capstone project course in systems or software engineering. USC has offered such a course since 1995, spanning and evolving across more than 200 real-client projects and 2000 students.

How Is the Book Organized?

The book generally flows from *why*, moves to *what*, and then on to *how*, with a bit of *how much* in between. It begins with a **Prologue**—a cautionary tale drawn from ancient mythology, but highly relevant to 21st-century system developers.

Once suitably enlightened, the reader will find a one-chapter **Introduction** describing our rationale for constructing the ICSM and a *high-level, self-contained overview* of ICSM fundamentals and use. System development stakeholders (e.g., users, developers, acquirers), executives, and managers may obtain a big-picture understanding of the ICSM, and find the summary to be food for thought and action in managing the uncertainties of modern complex product or system development. Readers who would prefer to start by exploring a particular aspect of the ICSM can generally use the Contents list or Index to find and address it in detail, but will often find it useful to refer back to the Introduction for overall context.

Part I provides detailed discussions of the *four key ICSM principles* and explains why they are critical. Each chapter in Part I begins with a failure story and a success story, illustrating the need for and application of the principle, followed by its key underlying practices. Part I completes the *why* part of the book begun in the Prologue and continued in the early part of the Introduction.

Parts II and III explain the *phases and stages* that provide the framework for ICSM's process generation. They introduce the *case study* that we use to illustrate how the stages and phases of the ICSM support success. This case study uses a next-generation medical device—an example of an advanced cyber–physical–human system with the inherent challenges of assuring safety, usability, and interoperability with other devices and systems—to lead the reader (and the medical device team) through the individual stages and phases of the ICSM. Parts II and III contain the majority of the *what* information, and a bit of the *how*.

Part IV completes the *how* and *how much* information. It supports implementation of the ICSM through *phase-combining patterns* and a set of *common cases* encountered in applying the risk-based phase decisions. There is information on adapting the ICSM to a specific project or environment, and an exploration of how its risk-driven, adaptive framework acts as a unifying element to support the effective application of existing practices. Part IV also provides guidance on applying some *key practices* that must be adapted somewhat for ICSM, and ends with an afterword that describes how we intend to evolve the ICSM with help from you, the reader.

The **Appendices** provide additional information on the *tools* developed specifically for ICSM activities, *mappings* of the ICSM to widely used process model and standards, and a comprehensive *bibliography*.

As stated earlier, the **Companion Website** to the book (http://csse.usc.edu/ICSM) provides the *EPG* and other *automated tools*, along with updates, examples, discussions, and *useful classroom materials*. The website is the primary place to find up-to-date information concerning the ICSM and its use, including white papers and guides for ICSM application in particular domains. While most of the material on the site is free, on occasion there may be material for sale. For those cases, the site is linked to and supported by Addison-Wesley and InformIT to provide an easy means to purchase those materials as well as other books of interest to the readers.

Who Helped Us Write the Book?

The organization and content of the ICSM have benefited significantly from our participation in three major efforts to provide improved guidelines for systems and software practice and education:

- The U.S. National Research Council's *Human–System Integration in the System Development Process* study
- The international efforts to define educational and practice guidelines that better integrate software, hardware, and human systems engineering—the *Graduate Software Engineering Reference Curriculum*
- The *Systems Engineering Body of Knowledge* and *Graduate Reference Curriculum for Systems Engineering*

These not only helped improve the ICSM, but also established its compatibility with these reference guidelines, along with co-evolving guidelines such as the IEEE-CS and ISO/IEC's *Software Engineering Body of Knowledge* and *INCOSE Systems Engineering Handbook*.

Funding for much of the initial work on the ICSM was provided through the Systems Engineering Research Center—a U.S. Department of Defense university-affiliated research center. In particular, Kristen Baldwin, Principal Deputy in the Office of the Deputy Assistant Secretary of Defense for Systems Engineering, provided early vision, guidance, and resources to the authors.

The following reviewers provided excellent advice and feedback on early versions of the book: Ove Armbrust, Tom DeMarco, Donald Firesmith, Tom Gilb, Paul Grünbacher, Liguo Huang, DeWitt Latimer IV, Bud Lawson, Jürgen Münch, George Rebovich, Jr., Neil Siegel, Hillary Sillitto, Qing Wang, Da Yang, and Wen Zhang.

The authors have gained numerous insights from collaborations and workshops with our Industrial Affiliate members, including:

Aerospace Corporation: Wanda Austin, Kirstie Bellman, Myron Hecht, Judy Kerner, Eberhardt Rechtin Marilee Wheaton

Agile Alliance: Kent Beck, Alistair Cockburn, Jim Highsmith, Ken Schwaber

AgileTek: John Manzo

AT&T: Larry Bernstein

BAE Systems: Jim Cain, Gan Wang

Bellcore: Stuart Glickman

Boeing: Ray Carnes, Marilynn Goo, Tim Peters, Shawn Rahmani, Bill Schoening, David Sharp

C-Bridge: Charles Leinbach

Cisco: Sunita Chulani, Steve Fraser

CMU-SEI: Roger Bate, Paul Clements, Steve Cross, Bill Curtis, Larry Druffel, John Goodenough, Watts Humphrey, Paul Nielsen

Construx, Inc.: Steve McConnell

Cubic Corporation: Mike Elcan

EDS: Mike Sweeney

Fraunhofer-IESE: Dieter Rombach

Fraunhofer-Maryland: Vic Basili, Forrest Shull, Marvin Zelkowitz

Galorath: Dan Galorath, Denton Tarbet

GE Systems: Paul Rook

General Dynamics: Michael Diaz

Group Systems: Bob Briggs

Hughes: Elliot Axelband

IBM/Rational: Tim Bohn, Grady Booch, Peter Haumer, Ivar Jacobson, Per Kroll, Bruce McIsaac, Philippe Kruchten, Walker Royce

Intelligent Systems: Azad Madni

ISCAS: Mingshu Li, Qing Wang, Ye Yang

ITT/Quanterion: Tom McGibbon

JPL: Jairus Hihn, Kenneth Meyer, Robert Tausworthe

Lockheed Martin: Sandy Friedenthal, John Gaffney, Gary Hafen, Garry Roedler

Master Systems: Stan Rifkin

Microsoft: Apurva Jain

MITRE: Judith Dahmann, George Rebovich

Motorola: Dave Dorenbos, Nancy Eickelmann, Arnold Pittler, Allan Willey

Naval Postgraduate School: Ray Madachy

NICTA: Ross Jeffery

Northrop Grumman/TRW: Frank Belz, George Friedman, Rick Hefner, Steve Jacobs, Alan Levin, Fred Manthey, Maria Penedo, Winston Royce, Rick Selby, Neil Siegel

OGR Systems: Kevin Forsberg

Price Systems: Arlene Minkiewicz, David Seaver

Raytheon: Anthony Peterson, Quentin Redman, John Rieff, Gary Thomas

RCI: Don Reifer

SAIC: Dick Fitzer, Tony Jordano, Beverly Kitaoka, Gabriel Lengua, Dick Stutzke

San Diego State University: Teresa Larsen

Softstar Systems: Dan Ligett

Software Metrics: Betsy Clark, Brad Clark

Stevens Institute: Art Pyster

Teledyne Brown Engineering: Douglas Smith

University of Massachusetts: Lori Clarke, Lee Osterweil

University of Texas: Dewayne Perry

University of Virginia: Kevin Sullivan

Wellpoint: Adam Kohl

Xerox: Peter Hantos, Jason Ho

Finally, the authors are grateful for the support of their partners in life, who put up with working weekends, late nights, unexpected travel, and all of the household inconveniences that writing books entail. To Sharla, Mike, Sohrab, and Jo—our best friends, greatest inspirations, sharpest critics, and truest loves—our heartfelt thanks. We love you.

About the Authors

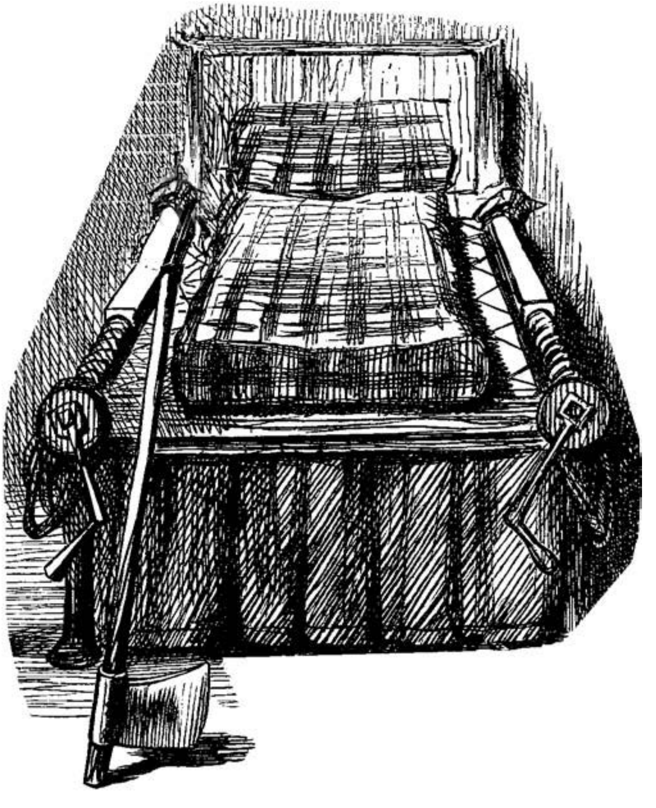
Barry Boehm developed a conceptual version of the spiral model at TRW in 1978, but only in 1981 was he able to employ it successfully, leading the development of a corporate TRW software development environment. Since the formal publication of this model in 1988, he and his colleagues have devoted extensive efforts to clarifying and evolving it through several intermediate versions into the ICSM. Dr. Boehm is the USC Distinguished Professor of Computer Sciences, Industrial and Systems Engineering, and Astronautics; the TRW Professor of Software Engineering; the Chief Scientist of the DoD–Stevens–USC Systems Engineering Research Center; and the Founding Director of the USC Center for Systems and Software Engineering. He was director of DARPA-ISTO for 1989–1992, at TRW for 1973–1989, at Rand Corporation for 1959–1973, and at General Dynamics for 1955–1959. Dr. Boehm is a Fellow of the primary professional societies in computing (ACM), aerospace (AIAA), electronics (IEEE), systems engineering (INCOSE), and lean and agile development (LSS), and a member of the U.S. National Academy of Engineering.

Jo Ann Lane is currently the systems engineering Co-Director of the University of Southern California Center for Systems and Software Engineering, a member of the Systems Engineering Research Center (SERC) Research Council representing the system of systems research area, and emeritus professor of computer science at San Diego State University. Her current areas of research include system of systems engineering, system affordability, expediting systems engineering, balancing lean and agile techniques with technical debt, and innovation in systems engineering. Previous publications include more than 50 journal articles and conference papers. In addition, Dr. Lane was co-author of the 2008 Department of Defense's *Systems Engineering Guide for Systems of Systems* and a contributor to the *Systems Engineering Body of Knowledge* (SEBoK). Prior to her current work in academia, she was a Vice President in SAIC's Healthcare and Software and Systems Integration groups.

Supannika Koolmanojwong is a faculty member and researcher at the University of Southern California Center for Systems and Software Engineering. Her primary research areas are systems and software process modeling, software process improvement, software process quality assurance, software metrics and measurement, agile and lean software development and expediting systems engineering. She is a certified ScrumMaster and a certified Product Owner. Prior to joining USC, Dr. Koolmanojwong was a software engineer and a RUP/OpenUp Content Developer at IBM RationalSoftware Group.

Richard Turner has more than 30 years of experience in systems, software, and acquisition engineering. He is currently a Distinguished Service Professor at the Stevens Institute of Technology in Hoboken, New Jersey, and a Principal Investigator with the Systems Engineering Research Center. Although on the author team for CMMI, Dr. Turner is now active in the agile, lean, and Kanban communities. He is currently studying agility and lean approaches as a means to solve large-systems issues. Dr. Turner is a member of the Executive Committee of the NDIA/AFEI Agile for Defense Adoption Proponent Team, is a member of the INCOSE Agile SE Working Group, and was an author of the groundbreaking IEEE Computer Society/PMI Software Extension for the Guide to the PMBOK that spans the gap between traditional and agile approaches. He is a Fellow of the Lean Systems Society, a Golden Core awardee of the IEEE Computer Society, and co-author of three other books: *Balancing Agility and Discipline: A Guide for the Perplexed*, co-written with Barry Boehm; *CMMI Survival Guide: Just Enough Process Improvement*, co-authored with Suzanne Garcia; and *CMMI Distilled*.

This page intentionally left blank



The Mythical Bed of Procrustes (with Tailoring Tools)

Prologue

A Cautionary Tale: The Bed of Procrustes

In the ancient world of the Greeks, there were gods and goddesses, demi-gods and heroes. The normal Greeks were quite entertained by the antics of these divine and semi-divine creatures, and followed them in their spare time (when they weren't creating democracy, mathematics, astronomy, history, and all manner of interesting things we occasionally use and appreciate today). There is a wealth of literature on the gods and goddesses, but we are interested in only one minor miscreant, who provides a wonderful metaphor for one of the main reasons this book was written.

His name was Procrustes, and he was a son of Poseidon, the god of the sea, among other things. Procrustes, although trained as a smith, made his living as an innkeeper cum bandit, having a nice hostelry on one of the mountains that happened to be on the way between two fairly important towns in ancient Greece. Of course, Procrustes wasn't your usual, run-of-the-mill bandit. Think of him as an early incarnation of a cross between Lizzy Borden and Norman Bates. While not someone you would want your sister to marry, he was creative in the way he relieved unlucky travelers of their goods. This creativity buys him a bit of mythological slack, as well as provides our metaphor.

Procrustes liked things to fit nicely into specified buckets—very much like many of the program managers and executives we have met along the way. He had an iron bed that he believed was the perfect length. In fact, he thought it should fit everyone. Procrustes did not have a therapist, so we'll probably never know the reason he was so enamored by the bed. Instead, we'll simply assume there are deep-seated reasons for his fixation, feel sorry for his affliction, and get on with the story.

His hostelry offered a night's rest for those who traveled the road across Mount Korydallos on the way between Athens and Eleusis. The stories are not clear as to how Procrustes selected his victims, but he would invite them in, show them his cherished bed, and offer it to them for the night, claiming, not unlike modern mattress salespeople, that it was magical and would perfectly fit whoever slept in it.

As statisticians and human factors experts will tell you, humans, even in the time of the ancient Greeks, generally varied in height and weight according to a normal distribution. And, of course, the iron bed was not created to adjust easily for such a distribution. In fact, it was a very precise length and width. It should be clear that the odds of having a person perfectly fit this bed, while not impossible, were probabilistically small. Ignoring the odds, or perhaps depending on them, Procrustes was nearly always presented with a person who did not fit the bed.*

Procrustes would bind the person to the bed, quickly realize that the guest did not fit it perfectly, reach for his smith's tools, and then carefully tailor the person to fit it—less magically, and more messily. If the unfortunate guest was too tall or too wide, he would simply lop off the offending parts. If too short or too narrow, then he would forcefully stretch the individual out until he fit. Needless to say, this generally proved fatal to the guest. Having assured himself of the perfection of the bed, and shaking his head at the imperfection of this particular human, Procrustes would gather the now-deceased's valuables into his hoard and begin the task of cleaning the room for his next guest.

Procrustes, whose name, ironically or mythically, meant "he who stretches," continued this endeavor until he mistakenly invited the hero Theseus to stay the night. Theseus turned the tables (or the bed, as it were) on Procrustes and did some tailoring of his own. While the disposition of Procrustes's famous bed is not reported, the concept of "one size fits all" has found its way down through the centuries.

The Point of the Story

Many organizations today find that their previous world of relatively stable businesses, products, processes, personnel, and technology is changing at an increasingly rapid pace. They find their investments in one-size-fits-all corporate and development processes are functioning like a Procrustean bed when applied to engineer and develop an increasing diversity of system types. They encounter problems with emergent and rapidly changing requirements and different balances of needs for agility, assurance, or both. The need for personnel with different skills, motivations, and lifestyles surfaces. Their rapidly evolving information and communication infrastructures are increasingly penetrating physical systems via three-dimensional printing and Internets of Things.

* In fact, some writers suggest that there were two beds, giving Procrustes even better odds.

Unfortunately, trying to escape from their Procrustean bed is difficult. There are conflicts between their impatient, change-oriented technical people and their settled, THWADI ("That's How We've Always Done It") administrators, each of whom has little understanding of the others' world. Employees working in single domains where one size is enough feel that *their* solutions ought to work for everybody else. It is even challenging to identify criteria for selecting alternative processes. The organization may have tried changing everyone to a new method and found that it is yet just another Procrustean bed.

We have gone through these difficulties ourselves during our periods in industry, government, and academia: trying to undo overenthusiastic corporate commitments made using the waterfall model; trying to get flexible acquisition standards approved by inflexible standards administrators; and trying to evolve best practices to teach students and have them apply in real-client project courses. The Incremental Commitment Spiral Model is the best approach we have found so far, and our applications of it across a wide range of project sizes and domains have worked out better than the project stakeholders' previous experiences. As we learn more, this model continues to evolve. We have also found that it is better to adopt its changes to organizations' current practices incrementally, and have identified practices that can be adopted incrementally, based on understanding organizations' strongest needs and opportunities.

We are not alone recognizing the problems. Other initiatives are making progress in moving people and organizations away from their previous one-size-fits-all processes. Several of our University of Southern California (USC) industrial affiliates have developed criteria for selecting alternative process models. Per Kroll and Philippe Kruchten's book, *The Rational Unified Process Made Easy*, separates its guidance into four tracks: Projects Deimos, Ganymede, Mars, and Jupiter. Frank Kendall's reorganization of the previously Procrustean U.S. Department of Defense Instruction 5000.02 into six different system acquisition swim lanes is another major step forward. We hope that this book and its website can benefit your organization and enable it to avoid having future projects stretched or lopped to fit Procrustean beds.

This page intentionally left blank

3

The Third Principle: Concurrent Multidiscipline Engineering

"Do everything in parallel, with frequent synchronizations."

—Michael Cusumano and Richard Selby, *Microsoft Secrets,* 1995

"As the correct solution of any problem depends primarily on a true understanding of what the problem really is, and wherein lies its difficulty, we may profitably pause upon the threshold of our subject to consider first, in a more general way, its real nature: the causes which impede sound practice; the conditions on which success or failure depends; the directions in which error is most to be feared. Thus we shall attain that great perspective for success in any work—a clear mental perspective, saving us from confusing the obvious with the important, and the obscure and remote with the unimportant."

—Arthur M. Wellington, *The Economic Theory of the Location of Railroads*, 1887

The first flowering of systems engineering as a formal discipline focused on the engineering of complex physical systems such as ships, aircraft, transportation systems, and logistics systems. The physical behavior of the systems could be well analyzed by mathematical techniques, with passengers treated along with baggage and merchandise as a class of logistical objects with average sizes, weights, and quantities. Such mathematical models were very good in analyzing the physical performance tradeoffs of complex system alternatives. They also served as the basis for the development of elegant mathematical theories of systems engineering.

The physical systems were generally stable, and were expected to have long useful lifetimes. Major fixes or recalls of fielded systems were very expensive, so it was worth investing significant up-front effort in getting their requirements to be complete, consistent, traceable, and testable, particularly if the development was to be contracted out to a choice of competing suppliers. It was important not to overly constrain the solution space, so the requirements were not to include design choices, and the design could not begin until the requirements were fully specified.

Various sequential process models were developed to support this approach, such as the diagonal waterfall model, the V-model (a waterfall with a bend upward in the middle), and the two-leg model (an inverted V-model). These were effective in developing numerous complex physical systems, and were codified into government and standards-body process standards. The manufacturing process of assembling physical components into subassemblies, assemblies, subsystems, and system products was reflected in functional-hierarchy design standards, integration and test standards, and work breakdown structure standards as the way to organize and manage the system definition and development.

The fundamental assumptions underlying this set of sequential processes, prespecified requirements, and functional-hierarchy product models began to be seriously undermined in the 1970s and 1980s. The increasing pace of change in technology, competition, organizations, and life in general made assumptions about stable, prespecifiable requirements unrealistic. The existence of cost-effective, competitive, incompatible commercial products or other reusable non-developmental items (NDIs) made it necessary to evaluate and often commit to solution components before finalizing the requirements (the consequences of not doing this will be seen in the failure case study in Chapter 4). The emergence of freely available graphic user interface (GUI) generators made rapid user interface prototyping feasible, but also made the prespecification of user interface requirement details unrealistic. The difficulty of adapting to rapid change with brittle, optimized, point-solution architectures generally made optimized first-article design to fixed requirements unrealistic.

As shown in the "hump diagram" of Figure 0-5 in the Introduction, the ICSM emphasizes the principle of concurrent rather than sequential work for understanding needs; envisioning opportunities; system scoping; system objectives and requirements determination; architecting and designing of the system and its hardware, software, and human elements; life-cycle planning; and development of feasibility evidence. Of course, the humps in Figure 0-5 are not a one-size-fits-all representation of every project's effort distribution. In practice, the evidence- and risk-based decision criteria discussed in Figures 0-7 and 0-8 in the Introduction can determine which specific process model will fit best for which specific situation. This includes situations in which the sequential process is still best, as its assumptions still hold in some situations. Also, since requirements increasingly emerge from use, working on all of the requirements and solutions in advance is not feasible—which is where the ICSM Principle 2 of incremental commitment applies.

This establishes the context for the "Do everything in parallel" quote at the beginning of this chapter. Even though preferred sequential-engineering situations still exist in which "Do everything in parallel" does not universally apply, it is generally best to apply it during the first ICSM Exploratory phase. By holistically and concurrently addressing during this beginning phase all of the system's hardware, software, human factors, and economic considerations (as described in the Wellington quote at the beginning of the chapter), projects will generally be able to determine their process drivers and best process approach for the rest of the system's life cycle. Moreover, as discussed previously, the increasing prevalence of process drivers such as emergence, dynamism, and NDI support will make concurrent approaches increasingly dominant.

Thus suitably qualified, we can proceed to the main content of Chapter 3. Our failure and success case studies are two different sequential and concurrent approaches to a representative complex cyber–physical–human government system acquisition involving remotely piloted vehicles (RPVs). The remaining sections will discuss best practices for concurrent cyber–physical–human factors engineering, concurrent requirements and solutions engineering, concurrent development and evolution engineering, and support of more rapid concurrent engineering.

An example to illustrate ICSM concurrent-engineering benefits is the unmanned aerial system (UAS; i.e., RPV) system enhancement discussed in Chapter 5 of the NRC's *Human–System Integration* report [1]. These RPVs are airplanes or helicopters operated remotely by humans. The systems are designed to keep humans out of harm's way. However, the current RPV systems are human-intensive, often requiring two people, and often considerably more, to operate a single vehicle. The increase in need to operate numerous RPVs is causing a strong desire to modify the 1:2 (one vehicle controlled by two people) ratio to allow for a single operator to operate more than one RPV, as shown in Figure 3-1.

A recent advanced technology demonstration of an autonomous-agent–based system enabled a single operator to control four RPVs flying in formation to a crisis area while compensating for changes in direction to avoid adverse weather conditions or no-fly zones. Often, such demonstrations to high-level decision makers, who are typically focused on rapidly getting innovations into the competition

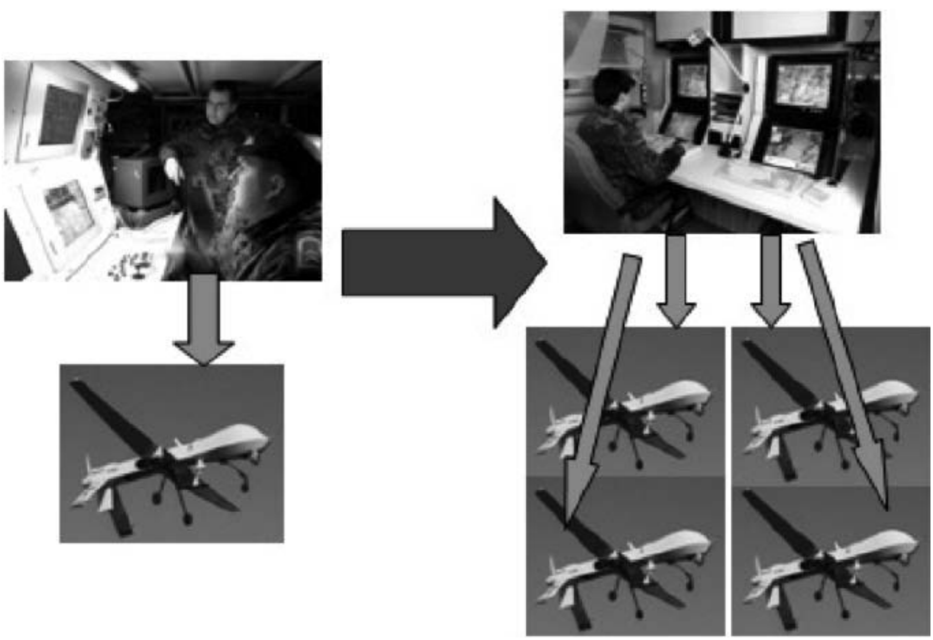


FIGURE 3-1 Vision of 4:1 Remotely Piloted Vehicle System (from Pew and Mavor, 2007)

space, will lead to commitments to major acquisitions before the technical and economic implications have been worked out (good examples have been the Iridium satellite-based personal telephone system and the London Ambulance System).

Based on our analyses of such failures and complementary successes (e.g., the rapid-delivery systems of Federal Express, Amazon, and Walmart), the failure and success stories in this chapter illustrate failure and success patterns in the RPV domain. In the future, the technical, economic, and safety challenges for similarly autonomous air vehicles will become even more complex, as with Amazon's recent concept and prototype of filling the air with tiny, fully autonomous, battery-powered helicopters rapidly delivering packages from its warehouse to your front door.

In this chapter, the demonstration of a 4:1 vehicle:controller ratio capability highly impressed senior leadership officials viewing the demo, and they established a high-priority rapid-development program to acquire and field a common agent-based 4:1 RPV control capability for use in battlefield-based, sea-based, and home-country–based RPV operations.

3.1 Failure Story: Sequential RPV Systems Engineering and Development

This section presents a hypothetical sequential approach representative of several recent government acquisition programs, which would use the demo results to create the requirements for a proposed program that used the agent-based technology to develop a 4:1 ratio system that enabled a single operator to control four RPVs in battlefield-based, sea-based, and home-country–based RPV operations. A number of assumptions were made to sell the program at an optimistic cost of $1 billion and schedule of 40 months. Enthusiasm was such that the program, budget, and schedule were established, and a multi-service working group of experienced battlefield-based, sea-based, and home-country–based RPV controllers was established to develop the requirements for the system.

The resulting requirements included the need to synthesize status information from multiple on-board and external sensors; to perform dynamic reallocation of RPVs to targets; to perform self-defense functions; to communicate status and observational information to central commanders and other RPV controllers; to control RPVs in the same family but with different releases having somewhat different controls; to avoid harming friendly forces or noncombatants; and to be network-ready with respect to self-identification when entering battle zones, establishing security credentials and protocols, operating in a publish–subscribe environment, and participating in replanning activities based on changing conditions. These requirements were included in a request for proposal (RFP) that was sent out to prospective bidders.

The winning bidder provided an even more impressive demo of agent technology and a proposal indicating that all of the problems were well understood, that a preliminary design review (PDR) could be held in 120 days, and that the cost would be only $800 million. The program managers and their upper management were delighted at the prospect of saving $200 million of the taxpayers' money, and they established a fixed-price contract to develop the 4:1 system to the requirements in the RFP in 40 months, with a System Functional Requirements Review (SFRR) in 60 days and a PDR in 120 days.

At the SFRR, the items reviewed were transcriptions and small elaborations of the requirements in the RFP. They did not include any functions for coordinating the capabilities, and included only sunny-day operational scenarios. There were no capabilities for recovering from outages in the network, from the loss of RPVs, or from incompatible sensor data, or for tailoring the controls to battlefield-based, sea-based, or home-country–based control equipment. The contractor indicated that it had hired some ex-RPV controllers who were busy putting such capabilities together.

However, at the PDR, the contractor could not show feasible solutions for several critical and commonly occurring scenarios, such as coping with network outages, missing RPVs, and inconsistent data; having the individual controllers coordinate with each other; performing self-defense functions; tailoring the controls to multiple equipment types; and satisfying various network-ready interoperability protocols. As has been experienced in practice [2], such capabilities are much needed and difficult to achieve.

Because the schedule was tight and the contractor had almost run out of systems engineering funds, management proposed to address the problems by using a "concurrent engineering" approach of having the programmers develop the software capabilities while the systems engineers were completing the detailed design of the hardware displays and controls. Having no other face-saving alternative to declaring the PDR to be a failure, the customers declared the PDR to be passed.

Actually, proceeding into development while completing the design is a pernicious misuse of the term "concurrent engineering," as there is not enough time to produce feasibility evidence and to synchronize and stabilize the numerous off-nominal approaches taken by the software developers and the hardware-detail designers. The situation becomes even worse when portions of the system are subcontracted to different organizations, which will often reuse existing assets in incompatible ways. The almost-certain result for large systems is one or more off-nominal architecture-breakers that require large amounts of rework and throwaway software to reconcile the inconsistent architectural decisions made by the self-fulfilling "hurry up and code, because we will have a lot of debugging to do" programmers. Figure 3-2 shows the results of such approaches for two large TRW projects, in which 80% of the rework resulted from the 20% of problem fixes resulting from critical off-nominal architecture-breakers [3].

As a result, after 40 months and $800 million in expenditures, some RPV control components were developed but were experiencing integration problems, and even after descoping the performance to a 1:1 operator:RPV ratio, several problems were still unresolved. For example, the hardware engineers used their traditional approach to defining interfaces in terms of message content (e.g., "The sensor data crossing an interface is defined in terms of the following units, dimensions,

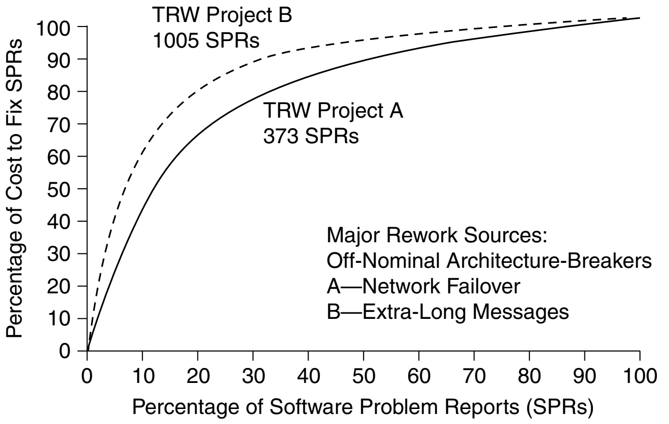


FIGURE 3-2 Results of Creating or Neglecting Off-Nominal Architecture-Breakers

coordinate systems, precision, frequency, or other characteristics"). They then took full earned value credit for defining the system's interfaces. However, the RPVs were operating in a Net-centric system of systems, where interface definition includes protocols for joining the network, performing security handshakes, publishing and subscribing to services, leaving the network, and so on. As there was no earned value left for defining these protocols, they remained undefined while the earned value system continued to indicate full credit for interface definition. The resulting rework and overruns could be said to result from off-nominal architecture breakers or from shortfalls in the concurrent engineering of the sensor data processing and networking aspects of the system, and from shortfalls in accountability for results.

Eventually, the 1:1 capability was achieved and the system delivered, but with reduced functionality, a cost of $3 billion, and a schedule of 80 months. Even worse, the hasty patching to get the first article delivered left the customer with a brittle, poorly documented, poorly tested system that would be the source of many expensive years of system ownership and sub-par performance.

3.2 Success Story: Concurrent Competitive-Prototyping RPV Systems Development

A concurrent incremental-commitment approach to the agent-based RPV control opportunity, using the ICSM process and competitive prototyping, would recognize that there were a number of risks and uncertainties involved in going from a single-scenario proof-of-principle demo to a fieldable system needing to operate in more complex scenarios. It would decide that it would be good to use prototyping as a way of buying information to reduce the risks, and would determine that a reasonable first step would be to invest $25 million in an Exploration phase. This would initially involve the customer and a set of independent experts developing operational scenarios and evaluation criteria from the requirements in Section 3.1 (to synthesize status information from multiple on-board and external sensors; to perform dynamic reallocation of RPVs to targets; to perform self-defense functions; and so on). These would involve not only the sunny-day use cases but also selected rainy-day use cases involving communications outages, disabled RPVs, and garbled data.

The customer would identify an RPV simulator that would be used in the competition, and would send out a request for information to prospective competitors to identify their qualifications to compete. Based on the responses, the customer would then select four bidders to develop virtual prototypes addressing the requirements, operational scenarios, and evaluation criteria, and providing evidence of their proposed agent-based RPV controllers' level of performance. The customer would then have the set of independent experts evaluate the bidders' results. Based on the results, it would perform an evidence- and risk-based Valuation Commitment Review to determine whether the technology was too immature to merit further current investment as an acquisition program, or whether the system performance, cost, and risk were acceptable for investing the next level of resources in addressing the problems identified and developing initial prototype physical capabilities.

As was discovered much more expensively in the failure case described earlier, the prospects for developing a 4:1 capability were clearly unrealistic. The competitors' desire to succeed led to several innovative approaches, but also to indications that having a single controller handle multiple-version RPV controls would lead to too many critical errors. Overall, however, the prospects for a 1:1 capability were sufficiently attractive to merit another level of investment, corresponding to a Valuation phase. This phase was funded at $75 million, some of the more ambitious key performance parameters were scaled back, the competitors were down-selected to three, and some basic-capability but multiple-version physical RPVs were provided for the competitors to control in several physical environments.

The evaluation of the resulting prototypes confirmed that the need to control multiple versions of the RPVs made anything higher than a 1:1 capability infeasible. However, the top two competitors provided sufficient evidence of a 1:1 system feasibility that a Foundations Commitment Review was passed, and $225 million was provided for a Foundations phase: $100 million for each of the top competitors, and $25 million for customer preparation activities and the independent experts' evaluations.

In this phase, the two competitors not only developed operational RPV versions, but also provided evidence of their ability to satisfy the key performance parameters and scenarios. In addition, they developed an ICSM Development Commitment Review package, including the proposed system's concept of operation, requirements, architecture, and plans, along with a Feasibility Evidence Description providing evidence that a system built to the architecture would satisfy the requirements and concept of operation, and be buildable within the budget and schedule in the plan.

The feasibility evidence included a few shortfalls, such as remaining uncertainties in the interface protocols with some interoperating systems, but each of these was covered by a risk mitigation plan in the winning competitor's submission. The resulting Development Commitment Review was passed, and the winner's proposed $675 million, 18-month, three-increment Stage II plan to develop an initial operational capability (IOC) was adopted. The resulting 1:1 IOC was delivered on budget and 2 months later than the original 40-month target, with a few lower-priority features deferred to later system increments. Figure 3-3 shows the comparative timelines for the Sequential and Concurrent approaches.

Of the $1 billion spent, $15 million was spent on the three discontinued Exploration-phase competitors, $40 million was spent on the two discontinued Valuation-phase competitors, and $100 million was spent on the discontinued Foundations-phase competitor. Overall, the competitive energy stimulated and the early risks avoided made this a good investment. However, the $125 million spent on the experience built up by the losing finalist could also be put to good use by awarding the finalist with a contract to build and operate a testbed for evaluating the RPV system's performance.

Actually, it would be best to announce such an outcome in advance, and to do extensive team building and award fee structuring to make the testbed activity constructive rather than adversarial.

While the sequential and concurrent cases were constructed in an RPV context from representative projects elsewhere, they show how a premature total commitment without adequate resources for and commitment to early concurrent engineering of the modeling, analysis, and feasibility assessment of the overall system will often lead to large overruns in cost and schedule, and performance that is

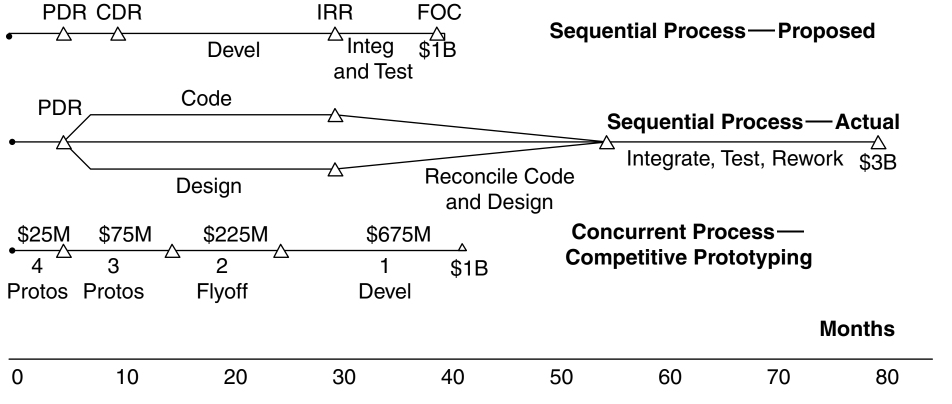


FIGURE 3-3 Comparative Timelines

considerably less than initially desired. However, by "buying information" early, the concurrent incremental commitment and competitive prototyping approach was able to develop a system with much less late rework than the sequential total commitment approach, and with much more visibility and control over the process.

The competitive prototyping approach spent about $155 million on unused prototypes, but the overall expenditure was only $1 billion as compared to $3 billion for the total-commitment approach, and the capability was delivered in 42 versus 80 months, which indicates a strong return on investment. Further, the funding organizations had realistic expectations of the outcome, so that a 1:1 capability was a successful realization of an expected outcome, rather than a disappointing shortfall from a promised 4:1 capability. In addition, the investment in the losing finalist could be put to good use by capitalizing on its experience to perform an IV&V role.

Competitive prototyping can lead to strong successes, but it is also important to indicate its potential failure modes. These include under-investments in prototype evaluation, leading to insufficient data for good decision making; extra expenses in keeping the prototype teams together and productive during often-overlong evaluation and decision periods; and choosing system developers too much on prototyping brilliance and too little on ability to systems-engineer and production-engineer the needed products [4]. These problem areas are easier to control in competitions among in-house design groups, where they are successfully used by a number of large corporations.

3.3 Concurrent Development and Evolution Engineering

As good as the success story in Section 3.2 appears to be, it could have a fatal flaw that is shared by many outsourced system acquisitions—namely, its primary focus on satisfying today's requirements as quickly and inexpensively as possible. This may build architectural decisions into the system that make it difficult to adapt to new opportunities or competitive threats. From an economic standpoint, this approach neglects the Iron Law of System Evolution:

> *For every dollar invested in developing a sustained-use system, be prepared to pay at least two dollars on the system's evolution.*

Data from hardware-intensive systems indicates that the average percentage of life-cycle cost spent on operations and support (O&S%) is a relatively small 12% for single-use consumables, but is 60% for ships, 78% for aircraft, and 84% for ground vehicles [5]. For software-intensive systems, O&S% figures from seven studies range from 60–70% to more than 90% [6].

Even so, many projects (and some system acquisition guidance documents) continue to emphasize such practices as "maximizing system performance while minimizing system acquisition costs." Such practices generally lead to brittle, point-solution architectures that overly constrain evolution options and inflate evolution costs, and to a lack of key system deliverables for reducing operations and support costs, such as maintenance and diagnostic tools and documentation, test case inputs and outputs, and latest-release COTS components. (COTS vendors generally support only their latest three releases. In one maintenance study, we encountered a system that was delivered with 120 COTS products, 66 of which were on releases that were no longer supported by the vendors.)

Several good practices for avoiding such situations can be applied in the initial ICSM Exploration phase. These include early addressing of post-deployment and aftermarket considerations such as development of a full operations concept description, including the following considerations:

- Identification and involvement of key operations and maintenance stakeholders
- Agreement on their roles and responsibilities
- Inclusion of total ownership costs in business case analyses
- Addressing of post-deployment supply chain management alternatives
- Identification of development practices and deliverables needed for successful operations and maintenance

Since operations and maintenance costs can consume 60% to 90% of an enterprise's resources, it is also important to build up a knowledge base on their nature, and to apply the knowledge to reduce their costs and difficulties. For example, this was done for the two TRW projects summarized in Figure 3-2. As indicated in Figure 3-2, their major sources of rework effort were found to be off-nominal architecture-breakers. This source of risk was added to the TRW risk management review guidelines for future projects. Also, their additional major sources of life-cycle change were determined to be hardware–software interfaces, new algorithms, subcontractor interfaces, user interfaces, external application interfaces, COTS upgrades, database restructuring, and diagnostic aids, as shown in Table 3-1.

Following Dave Parnas's information-hiding principles [7], these sources of change were encapsulated in the architectures of similar projects, and additional systems engineering effort was devoted to addressing off-nominal architecture breakers. As detailed in the next chapter, by investing more effort in systems engineering and architecting, the highly successful Command Center Processing and Display System-Replacement (CCPDS-R) system [8] flattened the usual exponential growth in cost to make changes even later in the life cycle. The resulting savings in total cost of ownership are shown in Figure 3-4 [9]. This figure indicates that the added investment in CCPDS-R was recouped via rework reduction by the end of the initial development cycle, and generated increasing savings in later cycles.

TABLE 3-1 Projects A and B Cost-to-Fix Data (Hours)

Category	Project A	Project B
Extra-long messages		3404 + 626 + 443 + 328 + 244 = 5045
Network failover	2050 + 470 + 360 + 160 = 3040	
Hardware-software interface	620 + 200 = 820	1629 + 513 + 289 + 232 + 166 = 2832
Encryption algorithms		1247 + 368 = 1615
Subcontractor interface	1100 + 760 + 200 = 2060	
GUI revision	980 + 730 + 420 + 240 + 180 = 2550	
Data compression algorithm		910
External applications interface	770 + 330 + 200 + 160 = 1460	
COTS upgrades	540 + 380 + 190 = 1110	741 + 302 + 221 + 197 = 1461
Database restructure	690 + 480 + 310 + 210 + 170 = 1860	
Routing algorithms	494 + 198 = 692	
Diagnostic aids	360	477 + 318 + 184 = 979
Total	**13,620**	**13,531**

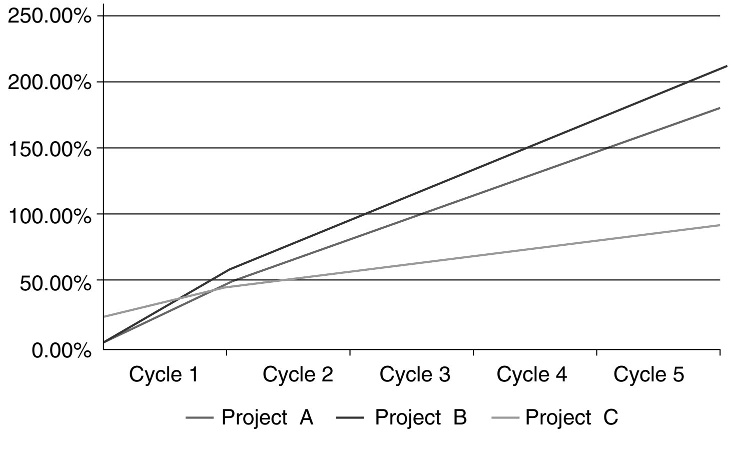


FIGURE 3-4 TOC's for Projects A, B, and C (CCPDS-R) Relative to Baseline Costs

3.4 Concurrent Engineering of Hardware, Software, and Human Factors Aspects

Not every system has all three hardware, software, and human factors aspects. When a system does have more than one of these aspects, however, it is important to address them concurrently rather than sequentially. A hardware-first approach will often choose best-of-breed hardware components with incompatible software or user interfaces; provide inadequate computational support for software growth; create a late software start and a high risk of a schedule overrun; or commit to a functional-hierarchy architecture that is incompatible with layered, service-oriented software and human-factors architectures [10].

Software-first approaches can similarly lead to architectural commitments or selection of best-of-breed components that are incompatible with preferred hardware architectures or make it hard to migrate to new hardware platforms (e.g., multiprocessor hardware components). They may also prompt developers to choose software-knows-best COTS products that create undesirable human–system interfaces. Human-factors-first approaches can often lead to the use of hardware–software packages that initially work well but are difficult to interoperate or scale to extensive use.

Other problems may arise from assumptions by performers in each of the three disciplines that their characteristics are alike, when in fact they are often very different. For systems having limited need or inability to modify the product once fielded (e.g., sealed batteries, satellites), the major sources of life-cycle cost in a hardware-intensive system are realized during development and manufacturing. However, as we noted earlier, hardware maintenance costs dominate (60–84% of life-cycle costs cited for ships, aircraft, and ground vehicles). For software-intensive systems, manufacturing costs are essentially zero. For information services, the range of 60% to 90% of the software life-cycle cost going into post-development maintenance and upgrades is generally applicable. For software embedded in hardware systems, the percentages would be more similar to those for ships and such. For human-intensive systems, the major costs are staffing and training, particularly for safety-critical systems requiring continuous 24/7 operations. A primary reason for this difference is indicated in rows 2 and 3 of Table 3-2. Particularly for widely dispersed hardware such as ships, submarines, satellites, and ground vehicles, making hardware changes across a fleet can be extremely difficult and expensive. As a result, many hardware deficiencies are handled via software or human workarounds that save money overall but shift the life-cycle costs toward the software and human parts of the system.

As can be seen when buying hardware such as cars or TVs, there is some choice of options, but they are generally limited. It is much easier to tailor software or human procedures to different classes of people or purposes. It is also much easier to deliver useful subsets of most software and human systems, while delivering a car without braking or steering capabilities is infeasible.

TABLE 3-2 Differences in Hardware, Software, and Human System Components

Difference Area	Hardware/ Physical	Software/Cyber/ Informational	Human Factors
Major life-cycle cost sources	Development; manufacturing; multilocation upgrades	Life-cycle evolution; low-cost multilocation upgrades	Training and operations labor
Nature of changes	Generally manual, labor-intensive, expensive	Generally straightforward except for software code rot, architecture-breakers	Very good, but dependent on performer knowledge and skills
Incremental development constraints	More inflexible lower limits	More flexible lower limits	Smaller increments easier, if infrequent
Underlying science	Physics, chemistry, continuous mathematics	Discrete mathematics, logic, linguistics	Physiology, behavioral sciences, economics
Testing	By test engineers; much analytic continuity	By test engineers; little analytic continuity	By representative users
Strengths	Creation of physical effects; durability; repeatability; speed of execution; 24/7 operation in wide range of environments; performance monitoring	Low-cost electronic distributed upgrades; flexibility and some adaptability; big-data handling, pattern recognition; multitasking and relocatability	Perceiving new patterns; generalization; guiding hypothesis formulation and test; ambiguity resolution; prioritizing during overloads; skills diversity
Weaknesses	Limited flexibility and adaptability; corrosion, wear, stress, fatigue; expensive distributed upgrades; product mismatches; human-developer shortfalls	Complexity, conformity, changeability, invisibility; common-sense reasoning; stress and fatigue effects; product mismatches; human-developer shortfalls	Relatively slow decision making; limited attention, concentration, multitasking, memory recall, and environmental conditions; teaming mismatches

The science underlying most of hardware engineering involves physics, chemistry, and continuous mathematics. This often leads to implicit assumptions about continuity, repeatability, and conservation of properties (mass, energy, momentum) that may be true for hardware but not true for software or human counterparts. An example is in testing. A hardware test engineer can generally count on covering a parameter space by sampling, under the assumption that the responses will be a continuous function of the input parameters. A software test engineer will have many discrete inputs, for which a successful test run provides no assurance that the neighboring test run will succeed. And for humans, the testing needs to be done by the operators and not test engineers.

A good example of integrated cyber–physical–human systems design is the detailed description of the Hospira medical infusion pump success story in Chapter 1. It included increasing risk-driven levels of detail in field studies and hardware–software–user interface prototyping; task analysis; hardware and software component analysis, including usability testing; and hardware–software–human safety analyses. Example prototypes and simulations included the following:

- Hardware industrial design mockups
- Early usability tests of hardware mockups
- Paper prototypes for GUIs with wireframes consisting of basic shapes for boxes, buttons, and other components
- GUI simulations using Flash animations
- Early usability tests with hardware mockups and embedded software that delivered the Flash animations to a touchscreen interface that was integrated into the hardware case

3.5 Concurrent Requirements and Solutions Engineering

With respect to the content of the Feasibility Evidence Description view of the ICSM in Figure 0-6 in the Introduction, the term "requirements" includes the definition of the system's operational concept and its requirements (the "what" and "how well" the system will perform). The term "solutions" includes the definition of the system–hardware–software–human factors architecture elements, and the project's plans, budgets, and schedules (the "how" and "how much").

For decades, and even today, standard definitions of corporate and government system development and acquisition processes have stipulated that the Requirements activity should produce complete, consistent, traceable, and testable requirements before any work was allowed on the solutions. Initially, there were some good reasons for this sequential approach. Often, requirements were inserted that were really solution choices, thus cutting off other solution choices that could have been much better. Or in many situations, developers would generate solutions before the requirements were fully defined or understood, leading to numerous useless features or misguided architectural commitments that led to large overruns. At the time, most systems were relatively simple and requirements were relatively stable, so that the risk of spending more time specifying them was less than the risk of expensive overruns.

However, the sequential requirements-first approach is a poor fit to most human approaches to practical problem solving. Figure 3-5 shows a representative result from a study of how people work when developing solutions, concurrently obtaining insights all the way from operational concepts to low-level solution components [11].

For more complex systems, teams of people will be similarly exploring and understanding multiple levels of problems and solutions and coordinating their

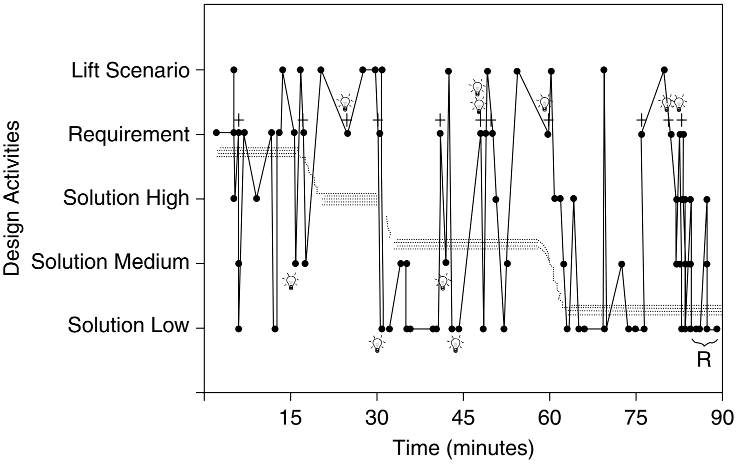


FIGURE 3-5 Human Problem Understanding and Solving: An Elevator (Lift) System Example

progress, capitalizing on many insights that are not available if they are locked into a sequential, reductionist, requirements-first approach. Also, they will have difficulties in developing key evidence such as business cases for the system, which require both estimates of system benefits (needing information about the requirements), and estimates of costs (needing information about the solutions).

Further, as systems become more complex and human-interactive, users become less able to specify their requirements in advance ("Which decision aids do I want to see on the computer screen or in the cockpit? I don't know, but I'll know it when I see it"—the IKIWISI syndrome). Also, as users gain experience in interactively using a system, new requirements emerge that may not be supportable by the architecture developed for the initial requirements (e.g., capabilities to cancel or undo commands, produce trend analyses, or decision outcome predictions).

Such hard-to-specify or emergent requirements are addressable via prototyping or solutions exploration, but these are not allowed in literal interpretations of sequential, requirements-first approaches, which tend to get ossified by layers of regulations, specifications, standards, contracting practices, and maturity models. One of the authors (Boehm) found himself in the difficult position of having led much of the effort to define the sequential, waterfall-oriented TRW Software Development Policies and Standards in the 1970s, along with training courses, review criteria, and corporate public relations materials—and then trying to convince projects in the 1980s to use counterculture techniques such as human-interface prototyping ("Prototyping is not allowed. It's developing solutions before we fully define the requirements").

The ICSM's principles and practices such as evidence- and risk-driven decision making provide ways to evolve to concurrent versus sequential requirements and solutions engineering. These considerations will be covered in the next chapter. Also, further details such as evidence-based process guidance are covered in Chapter 13. In addition, methods, processes, and tools for concurrent-engineering risk assessment and award-fee contracting are provided on the ICSM website at http://csse.usc.edu/ICSM.

References

[1] Pew, R., and Mavor, A. *Human–System Integration in the System Development Process.* NAS Press, 2007.

[2] Beidel, E. "Efforts Under Way to Harden Unpiloted Aircraft for Contested Airspace." *National Defense.* July 2011.

[3] Boehm, B., Valerdi, R., and Honour, E. "The ROI of Systems Engineering: Some Quantitative Results for Software-Intensive Systems." *Systems Engineering.* 2008;11(3):221–234.

[4] Ingold, D. "Results of a Survey on Competitive Prototyping for Software-Intensive Systems." USC-CSSE Technical Report USC-CSSE-2008-841. October 2008. http://csse.usc.edu/csse/TECHRPTS/2008/2008_main.html.

[5] Redman, Q. *Weapon System Design Using Life Cycle Costs.* Raytheon Presentation, NDIA, 2008.

[6] Koskinen, J. "Software Maintenance Fundamentals." In P. Laplante (Ed.), *Encyclopedia of Software Engineering.* Taylor & Francis Group, 2009.

[7] Parnas, D. "Designing Software for Ease of Extension and Contraction." *IEEE Transactions in Software Engineering.* March 1979;128–137.

[8] Royce, W. *Software Project Management: A Unified Framework.* Reading, MA: Addison-Wesley Professional, 1998.

[9] Boehm, B., Lane, J., and Madachy, R. "Total Ownership Cost Models for Valuing System Flexibility." *Proceedings of CSER 2011.* March 2011.

[10] Maier, M. "System and Software Architecture Reconciliation." *Systems Engineering.* 2006;9(2):146–159.

[11] Guindon, R. "Designing the Design Process: Exploring Opportunistic Thoughts." *Human–Computer Interaction.* 1990;5.

This page intentionally left blank

Index

F

G

H

Q

R