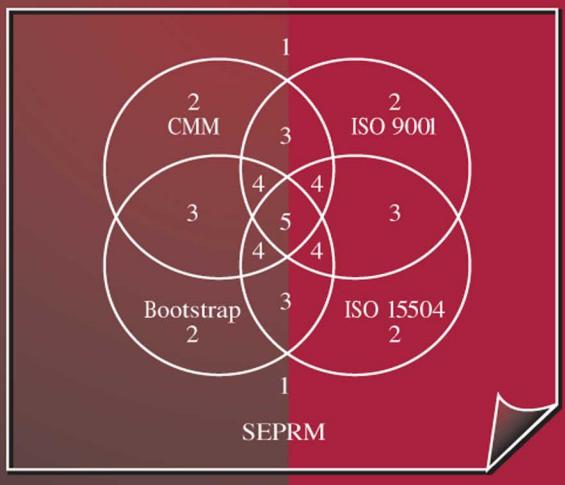
SOFTWARE ENGINEERING PROCESSES

Principles and Applications



Yingxu Wang
Graham King

SOFTWARE ENGINEERING PROCESSES

Principles and Applications

Yingxu Wang, Prof., Ph.D.
Graham King, Prof., Ph.D.

CRC Press
Boca Raton London New York Washington, D.C.

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Version Date: 20140501

International Standard Book Number-13: 978-1-4822-7454-7 (eBook - PDF)

To MY PARENTS Y.W.
To MY WIFE AND DAUGHTERS G.A.K.

" Deal with the difficult
while it is still easy.
Solve large problems
when they are still small.
Preventing large problems
by taking small steps
is easier than solving them.
By small actions
great things are accomplished."

– **Lao Tzu** (604-531 BC)

"Then there is a metatheoretic level where you study the whole structure of a class of problems. This is the point of view that we have inherited from logic and computability theory."

– **Richard Karp** (Turing Award Winner, 1985)

This page intentionally left blank

Contents Summary

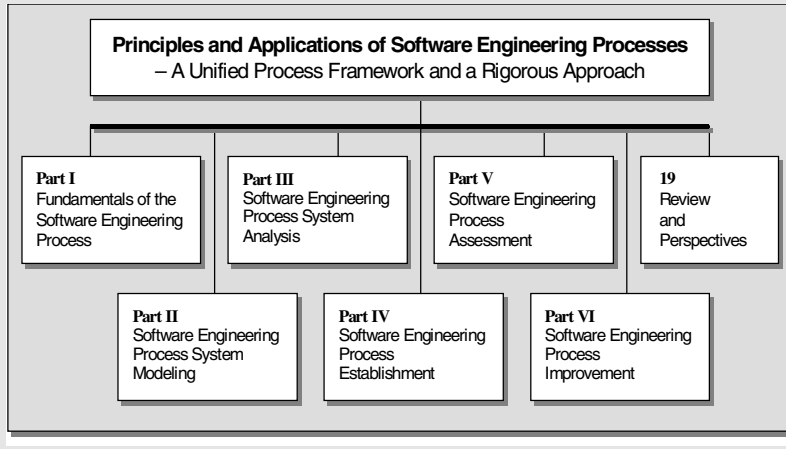


Table of Contents

This page intentionally left blank

Preface

Software engineering is a discipline of increasing importance in computing and informatics. The nature of problems in software engineering arise from the inherent complexity and diversity, the difficulty of establishing and stabilizing requirements, the changeability or malleability of software, the abstraction and intangibility of software products, the requirement of varying problem domain knowledge, the nondeterministic and polysolvability in design, the polyglotics and polymorphism in implementation, and the interactive dependency of software, hardware, and human being.

A new approach for dealing with the difficulties of large-scale software development emerged in the last decade. It sought to establish an appropriate software engineering process system. A software engineering process system is a set of empirical and best practices in software development, organization, and management which serves as an reference model for regulating the process activities in a software development organization.

Research into the software engineering process is a natural extension of scope from that of the software development methodologies necessary to meet the requirement for engineering large-scale software development. Conventional software development methodology studies cover methods, models, approaches, and phases of software development. The software engineering process, then, covers not only software development methodologies but also engineering methodologies and infrastructures for software corporation organization and project management.

As the scale of software increases continually and at an ever faster rate, greater complexity and professional practices become critical. Software development is no longer solely a black art or laboratory activity; instead, it has moved inexorably toward a key industrialized engineering process. In software engineering, the central role is no longer that of the programmers;

project managers and corporate management have critical roles to play. As programmers use programming technologies, software corporation managers seek organizational and strategic management methodologies, and project managers seek professional management and software quality assurance methodologies. These developments have resulted in a modern, expanded domain of software engineering which includes three important aspects: development methodology, organization and infrastructure, and management.

Understanding the need to examine the software engineering process follows naturally from the premise that has been found to be true in other engineering disciplines, that is, that better products result from better processes. For the expanded domain of software engineering, the existing methodologies that cover individual subdomains are becoming inadequate. Therefore, an overarching approach is sought for a suitable theoretical and practical infrastructure to accommodate all the modern software engineering practices and requirements. An interesting approach, which is capable of accommodating the complete domain of software engineering, has been recognized and termed the "software engineering process". Research into and adoption of the software engineering process paradigm will encompass all the approaches to software engineering.

To model the software engineering processes, a number of software process system models such as CMM, ISO 9001, BOOTSTRAP, ISO/IEC 15504 (SPICE) have been developed in the last decade. The variety and proliferation of software engineering process research and practices characterize the software engineering process as a young subdiscipline of software engineering that still needs integration and fundamental research. Studies in the software process reflect a current trend that shifts from controlling the quality of the final software product to the optimization of the processes that produce the software. It is also understood that the software engineering process, rather than the software products themselves, can be well established, stabilized, reused, and standardized.

A comprehensive and rigorous textbook is needed to address the unified and integrated principles, foundations, theories, frameworks, and best practices in software engineering process establishment, assessment, and improvement. This book is the first textbook intending to address both practical methodologies for process-based software engineering and the fundamental theories and philosophies behind them. This book covers broad areas of the new discipline of process-based software engineering such as software process foundations, modeling, analysis, establishment, assessment, and improvement.

The Aims of this Book

This book has emphasis on establishing a unified software engineering process framework integrating current process models, and developing a rigorous approach to process-based software engineering.

The aim is to investigate the philosophical, mathematical, and managerial foundations of software engineering; to establish a unified theoretical foundation for software engineering process modeling, analysis, establishment, assessment, and improvement; to explore the feature, orientation, interrelationship, and transformability of current process models; and to integrate the current process models and methodologies into a unified process framework–the software engineering process reference model (SEPRM).

Using a unified process framework, important current software process models are analyzed comparatively. The process frameworks and process capability determination methods of current models are formally described. As a means of introducing engineering rigor to process modeling, the algorithms of current process models are formally elicited. The ideas for mutual comparison between current models, the avoidance of ambiguity in application, and the simplification of manipulation for practitioners are addressed systematically.

The SEPRM is developed to show how to solve the problems of different process domains, orientations, structures, taxonomies, and methods. A set of process benchmarks has been derived that are based on a series of worldwide surveys of software engineering process practices. Based on the overarching SEPRM model and the unified process theory, this book demonstrates that for the first time, current process models can be integrated and their assessment results can be transformed from one to another. This has practical significance for those who are facing a requirement of multiple certifications, or to those who are pondering the implications of capability levels in different process models.

The Features of this Book

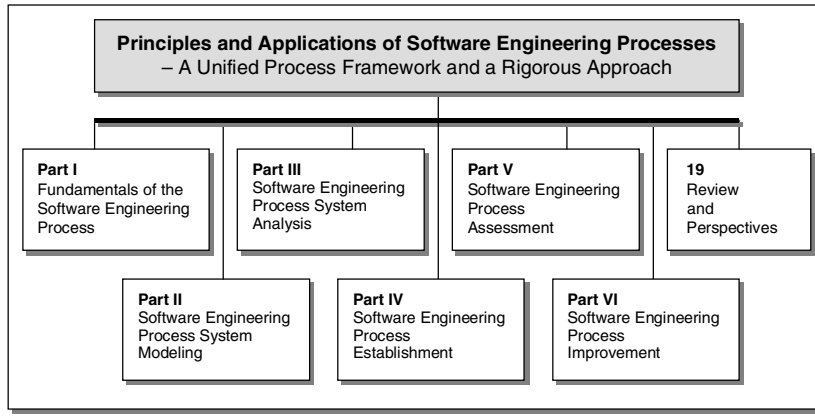
This book is characterized both as a comprehensive reference text for practitioners and as a *vade mecum* for students. The features of this book are that it:

- Investigates the philosophical, mathematical, and managerial foundations of software engineering
- Provides a unified software engineering process framework and an overarching software engineering process reference model (SEPRM)
- Develops a rigorous and practical approach to process-based software engineering
- Furnishes a detailed guide and case studies for practitioners in the industry
- Summarizes research findings, new methodologies, and applications in the discipline of software process engineering

Supplementary to the main body of text presented in this book, a number of reading aids is specially prepared for readers. A brief description of purposes is provided in front of each chapter, with a discussion of background of the chapter and its relationships to other parts and chapters of the book. A brief summary and a sidebar of knowledge structure are developed for each chapter which extract key knowledge and major achievements in each chapter. Annotated references are provided, helping readers to find related knowledge and/or alternative approaches in the literature, and to get familiar with the research and practices in the entire discipline of process-based software engineering.

The Structure of this Book

Software engineering process systems provide a fundamental infrastructure for organizing and implementing software engineering. The software engineering process discipline studies theories and foundations, modeling, analysis, establishment, assessment, improvement, and standardization of software processes. Viewing the knowledge structure of the software engineering process discipline as shown below, this book explores and investigates each topic systematically:



Part I – Fundamentals of the Software Engineering Process

The first part of this book investigates fundamentals of the software engineering process, and explores basic theories and empirical practices developed in this new discipline. A systematic and rigorous approach is taken in order to build a unified software engineering process framework.

The knowledge structure of this part, fundamentals of the software engineering process systems, is as follows:

- Chapter 1. Introduction
- Chapter 2. A Unified Framework of Software Engineering Process
- Chapter 3. Process Algebra
- Chapter 4. Process-Based Software Engineering

Based on the fundamental studies developed in this part, we will reach a key conclusion that software engineering is naturally a process system, and perhaps the most unique and complicated process system in all existing engineering disciplines.

Part II – Software Engineering Process System Modeling

Software engineering process system modeling explores a complete domain of software engineering processes, its architectures, and the fundamental framework. This part investigates current process models and contrasts them with the unified process framework developed in Part I. Comparative analyses of their interrelationships will be presented in Part III.

The knowledge structure of this part, software engineering process system modeling, is as follows:

- Chapter 5. The CMM Model
- Chapter 6. The ISO 9001 Model
- Chapter 7. The BOOTSTRAP Model
- Chapter 8. The ISO/IEC 15504 (SPICE) Model
- Chapter 9. The Software Engineering Process Reference Model: SEPRM

Part II adopts the unified process system framework developed in Part I as a fundamental common architecture in presenting the current process system models. A key conclusion of this part is that SEPRM is a superset paradigm of the current process models and the unified software engineering process framework. It is also demonstrated that the current process models can be fit within the unified framework of the software engineering process systems.

Part III – Software Engineering Process System Analysis

One of the most frequently-asked questions in the software industry is "what are the interrelationships between current process models?" In the previous part we presented formal views on individual process models. Part III explores the interrelationships between them via quantitative analysis, and investigates practical foundations of the software engineering process via benchmarking.

The knowledge structure of this part, software engineering process system analysis, is as follows:

- Chapter 10. Benchmarking the SEPRM Processes
- Chapter 11. Comparative Analysis of Current Process Models
- Chapter 12. Transformation of Capability Levels between Current Process Models

A rigorous and quantitative approach is adopted in order to analyze the characteristic attributes of process, the compatibility and correlation of process models, and the interrelationships and transformability of capability levels in different process models. Objective views on features and orientations of current process models are obtained from the analyses.

Part IV – Software Engineering Process Establishment

Software engineering process system establishment is the first important step in process-based software engineering because reliance is placed on both process assessment and improvement theories and practices. Working on the common foundation of a systematically established process system,

this part explores methodologies and approaches to software engineering process system establishment such as process model reuse, tailoring, extension, and adaptation. The relationships of these methodologies with the theories and unified process framework developed in previous parts are discussed. Examples and case studies such as a parallel process model for software quality assurance, a minimum process model for software project management, a tailored CMM model, and an extension of ISO/IEC TR 15504 model are provided for demonstrating the applications of the process establishment methodologies.

The knowledge structure of this part, software engineering process establishment, is as follows:

- Chapter 13. Software Process Establishment Methodologies
- Chapter 14. An Extension of ISO/IEC TR 15504 Model

In this part we adopt a pragmatic view on software engineering process system establishment, assessment, and improvement. Systematic process establishment is recognized as the foundation for process assessment and improvement. A software engineering process system reference model, SEPRM, is viewed as the central infrastructure for process-based software engineering.

Part V – Software Engineering Process Assessment

Here we explore how the theories and algorithms of process assessment developed in Part I and Part II are applied in real world process system assessments. Three practical process assessment methodologies, the model-based, the benchmark-based, and the template-based, are developed. These assessment methodologies provide a step-by-step guide to carry out a process assessment, and show the applications of the unified software engineering process framework and SEPRM in the software industry.

The knowledge structure of this part, software engineering process assessment, is as follows:

- Chapter 15. Software Process Assessment Methodologies
- Chapter 16. Supporting Tools for Software Process Assessment

In this part process assessment is recognized as the basic measure for process improvement. Software process system assessment methodologies are presented in a phase-by-phase and step-by-step manner. Especially, a set of practical templates is developed to support an assessment according to the SEPRM reference model.

Part VI – Software Engineering Process Improvement

Part VI examines philosophies and generic approaches to software engineering process improvement. Three process improvement methodologies, the model-based, the benchmark-based, and the template-based, are developed. These improvement methodologies provide step-by-step guides on how to carry out a process improvement in accordance with the SEPRM process framework and methodologies. A set of case studies of real-world process improvement is provided, and key successful factors and benefits in process improvement are analyzed. Roles, prerequirements, and techniques of software process improvement that provide a useful guide for implementing process improvement according to the SEPRM reference model are described.

The knowledge structure of this part, software engineering process improvement, is as follows:

- Chapter 17. Software Process Improvement Methodologies
- Chapter 18. Case Studies in Software Process Improvement

These chapters recognize process improvement as a complicated, systematic, and highly professional activity in software engineering that requires theory and models, skilled technical and managerial staff, and motivated top management commitment. A system engineering perception on software process improvement is adopted rather than the all-too-prevalent philosophy of "fire-fighting". A new approach of benchmark-based process improvement is developed based on the philosophy that not all processes at Level 5 are the best and most economic solutions in process improvement. Instead, a software organization may play its hand so that its process capability is aimed at an optimizing profile which is better than that of its competitors in the same area.

In conclusion Chapter 19 presents a review and perspectives on the discipline of software engineering in general, and the software engineering process in particular.

The Audience for this Book

The readership of this book is intended to include graduate, senior-level undergraduate students, and teachers in software engineering or computer science; researchers and practitioners in software engineering; and software engineers and software project and organization managers in the software industry.

This book provides a comprehensive and rigorous text addressing unified and integrated principles, foundations, theories, frameworks, methodologies, best practices, alternative solutions, open issues for further research, and plentiful resources in software engineering process establishment, assessment, and improvement. Readers in the following categories will find the book adds value to their work and pursuits:

- Software corporation executives seeking strategic solutions in software engineering and wishing to avoid not seeing the forest for the trees
- Software project managers seeking cutting-edge technologies, best practices, and practical aids for improving process capabilities
- Software engineers and practitioners seeking empirical process repositories and classical process paradigms, and who want to optimize their roles in the software engineering process systems
- Software engineering researchers seeking state-of-the-art theories, approaches and methodologies, representative process paradigms, and open issues for further studies in software engineering and software engineering processes
- Teachers and trainers in software engineering seeking a systematic textbook on principles and applications of software engineering processes with a unified theoretical framework, comparative and critical analyses, a well-organized body of knowledge, in-depth comments, and questions and answers (in separate volumes)
- Students and trainees in software engineering seeking a systematic textbook providing academic views, clear knowledge structures, critical analyses, and plentiful annotated references
- System analysts seeking an insight into current process models and standards and their strengths and weaknesses; and wishing to mine a plentiful set of data surveyed in the software industry
- Software process assessors seeking theoretical and empirical guides, relationships and process capability transformation between current process models, as well as practical templates and supporting tools
- Software tool developers seeking an insight into process system framework structures, methodologies, algorithms, and interrelationships

This book is self-contained and only basic programming experience and software engineering concepts are required. This book is designed and expected to appeal to developers, scholars, and managers because software engineering methodologies and software quality issues are leading the agenda in the light of the information era.

Acknowledgments

This work was carried out in collaboration with the ISO/IEC JTC1/SC7, IEEE Technical Council on Software Engineering (TCSE) and Software Engineering Standard Committee (SESC), European Software Institute, the BSI Expert Panel ITS15-400 on Software Engineering, BCS, and IBM (Europe). The authors would like to acknowledge their support.

The authors sincerely thank our colleagues Ian Court, Margaret Ross, Geoff Staples, Alec Dorling, and Antony Bryant for many enjoyable discussions, debates, and proofreadings. We would like to acknowledge the inspiration from the work of C.A.R. Hoare, Watts Humphrey, Victor Basili, Barry Boehm, David L. Parnas, Jeff Kramer, Ian Sommerville, Manny Lehman, Wilhelm Schafer, Geoff Dromey, Ali Mili, Terry Rout, Mark Paulk, David Kitson, Khaled El Emam, Richard Messnarz, Pasi Kuvaja, Taz Duaghtrey, and Dilip Patel.

The authors would also like to thank the professional advice, practical assistance, or valuable help of Ron Powers, William Heyward, Dawn Mesa, Saba Zamir, Dawn Sullivan, Suzanne Lassandro, Jan-Crister Persson, Paula Kökeritz, Håkan Wickberg, Christine King, Huiling Yang, and Siyuan Wang.

Yingxu Wang
Graham King

Trademarks and Service Marks

BootCheck is a trademark of the BOOTSTRAP Institute

BOOTSTRAP is a service mark of the BOOTSTRAP Institute

CMM is a service mark of SEI

ISO 9000 is an international standard of ISO

ISO 9001 is an international standard of ISO

ISO/IEC 12207 is an international standard of ISO and IEC

ISO/IEC CD 15288 is a draft international standard of ISO and IEC

ISO/IEC TR 15504 is an international standard of ISO and IEC

PSP is a service mark of SEI

PULSE is a service mark of the PULSE Consortium

SEPRM (the unified Software Engineering Process Reference Model) is a service mark owned by the authors

TRILLIUM is a trademark of Northern Bell

TSP is a service mark of SEI

About the Authors

Yingxu Wang is Professor of Computer Science, and project manager with the Center for Software Engineering at IVF, Gothenburg, Sweden. He was a visiting professor in Computing Laboratory at Oxford University during 1995. He was awarded a PhD in software engineering by The Nottingham Trent University / Southampton Institute, UK.

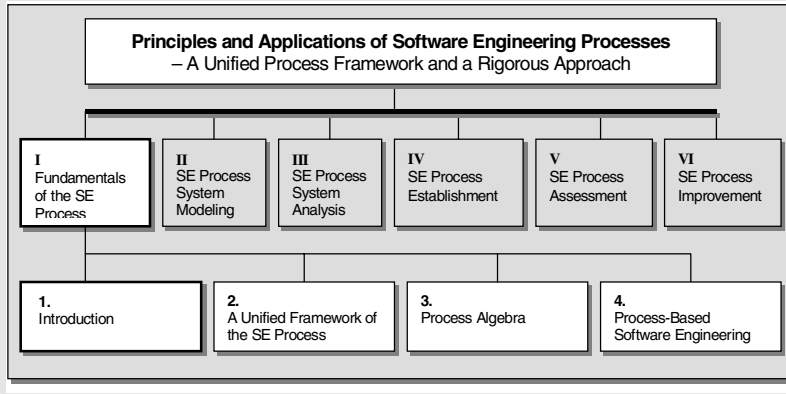
Dr. Wang is a member of IEEE TCSE/SESC, ACM, and ISO/IEC JTC1/SC7, and is Chairman of the Computer Chapter of IEEE Sweden. He has accomplished a number of European, Swedish, and industry-funded research projects as manager and/or principal investigator, and has published over 100 papers in software engineering and computer science. He has won a dozen research achievement and academic teaching awards in the last 20 years. He can be reached at Yingxu.Wang@acm.org, Y.X.Wang@ieee.org, or http://msnhomepages.talkcity.com/CerfSt/DrYWang/ .

Graham King is Professor of Computer Systems Engineering at Southampton Institute, UK. He heads a research center which has a strong interest in all aspects of software engineering. He was awarded a PhD in architectures for signal processing by the Nottingham Trent University and has authored two previous books, and nearly 100 academic papers.

Dr. King is a member of the British Computer Society and the IEEE, and has been principal investigator for a number of research council and industry-funded projects. He can be reached at king_g@ieee.org.

PART I

FUNDAMENTALS OF THE SOFTWARE ENGINEERING PROCESS



The software engineering process is a set of sequential activities for software engineering organization, implementation, and management, which are functionally coherent and reusable. The first part of this book investigates fundamentals of the software engineering process and explores basic theories and empirical practices developed in this new discipline.

The knowledge structure of this part is as follows:

- Chapter 1. Introduction – The Nature of Software Engineering
- Chapter 2. A Unified Framework of the Software Engineering Process
- Chapter 3. Process Algebra
- Chapter 4. Process-Based Software Engineering

This part sets up the basic theories, a systematic and rigorous approach to the building of a unified software engineering process framework, and a set of defined and consistent terminology for the discipline of software engineering process and the practice of process-based software engineering.

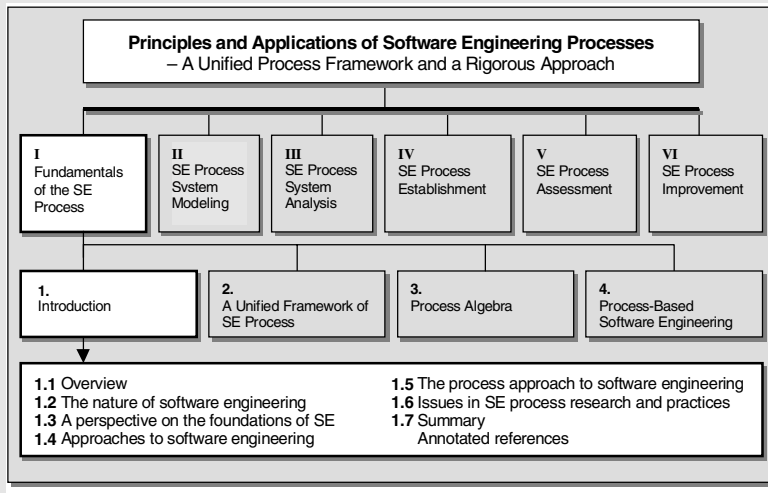
Chapter 1 explores the philosophical, mathematical, and managerial foundations of software engineering. Chapter 2 establishes the theoretical foundations of process-based software engineering through investigating the generic software development organization model, process model, capability model, capability determination method, and process assessment/ improvement methods in software engineering. A formal process notation system, process algebra, is introduced in Chapter 3, followed by Chapter 4, which provides a generic view of methodologies of process-based software engineering.

The unified process theories and framework will be applied in the remaining parts of the book. The fundamental process theories will also be used as guidelines to organize the empirical best practices and processes that are found useful in the software industry.

Based on the fundamental studies developed in this part, we will reach a key conclusion that software engineering is perhaps the most unique discipline and the most complicated process system in almost all existing engineering disciplines.

Chapter 1

INTRODUCTION



1.1 Overview

Software engineering is an increasingly important discipline that studies large-scale software development methodologies and approaches. Recently, software engineering has shifted from a laboratory-oriented profession to a more industry-oriented process. This trend reflects the needs of the software industry moving toward integrating software development techniques with organization and management methodologies to form a process-based software engineering environment.

Historically, software engineering has focused on programming methodologies, programming languages, software development models, and tools. From the domain coverage point of view, these approaches have concentrated on purely technical aspects of software engineering. Areas now thought critical to software engineering – organizational and management infrastructures – have been largely ignored.

As software systems increase in scale, issues of complexity and professional practices become critical. Software development is no longer solely an academic or laboratory activity; instead, it has become a key industrialized process. In the software industry, the central role is no longer that of the programmers because project managers and corporate management also have critical roles to play. As programmers require programming technologies, the software corporation managers seek organization and strategic management methodologies, and the project managers seek management and software quality assurance methodologies. These needs have together formed the modern domain of software engineering which, to summarize, includes three important aspects: development methodology and infrastructure, organization, and management.

For this expanded domain of software engineering, the existing methodologies that cover individual subdomains are becoming inadequate. Therefore, an overarching approach is sought for a suitable theoretical and practical infrastructure in order to accommodate the full range of modern software engineering practices and requirements. An interesting approach, which is capable of subsuming most of these domains of software engineering, is the process-based software engineering methodology. Research into, and adoption of, the software engineering process approach may be made to encompass all the existing approaches to software engineering.

In defining a software engineering process system it is natural to think of a set of empirical practices in software development, organization, and management which comprises an abstract model of the entire set of activities within software development organizations. It is a case of standing back and seeing the bigger picture. Over and above the traditional aspects such as methods, models, approaches, and phases of software development, the software engineering process also covers engineering methodologies suitable for large-scale software development, organization, and management.

To model the software engineering processes, a number of software engineering process system models such as CMM, ISO 9001, BOOTSTRAP, and ISO/IEC TR 15504 (SPICE) have been developed in the last decade. Hereafter, a software engineering process system model will be shortly referred to as a process model. It is noteworthy that the term process model is different from the conventional lifecycle model. The latter consists of "phases" and is oriented on software development while the former consists of processes and covers all practices in large-scale software project organization, management, and development.

The variety and proliferation in software engineering process research and practices characterize the software engineering process an emerging discipline in software engineering that still needs integration and fundamental study. Studies in the software process reflect a current trend [Humphrey, 1995; Kugler and Rementeria, 1995; and Pfleeger, 1998] that shifts from controlling the quality of the final software product to the optimization of the processes that produce the software. It is also understood that, although almost all application software development are one-off projects, the software engineering process can be well-established, stabilized, reused, and standardized.

This book aims to investigate the philosophical, mathematical, and managerial foundations of software engineering. It intends to establish a unified theoretical foundation for software engineering process modeling, analysis, establishment, assessment, and improvement, and to explore the orientation, interrelationship, and transformability of current process models. It demonstrates one way forward in the drive to integrate the current process models and methodologies into a super reference model.

To achieve this, current software engineering process models are comparatively analyzed. Their process frameworks and process capability determination methods are rigorously described, and algorithms are formally elicited. This rigorous approach enables the mutual comparison between current process models, the avoidance of ambiguity in description, and the simplification of manipulation in applications.

A software engineering process reference model (SEPRM) is developed [Wang et al., 1996a/97a/98a/99e] for using as a vehicle with which to solve

the problems of different process-domains, orientations, structures, frameworks and methods. A set of process benchmarks has been derived that is based on a series of worldwide surveys of software process practices [Wang et al., 1998a/99c], and these are used to support validation of the SEPRM model. Using the SEPRM approach, current process models can be integrated and their assessment results can be transformed from one to another for the first time.

In this chapter, the theoretical foundations of software engineering are explored. Then, existing approaches to software engineering in general and the process approach in particular are investigated. Based on the examination of the problems identified in software process research and practices, a systematic and algorithmic approach to software engineering process system modeling, analysis, establishment, assessment, and improvement is introduced.

1.2 The Nature of Software Engineering

The term software engineering was first reported in a European conference in 1968 [Naur and Randell, 1969; Bauer, 1976]. In this conference, Fritz Bauer introduced software engineering as:

> The establishment and use of sound engineering principles in order to obtain economical software that is reliable and works efficiently on real machines.

Since then, research on software engineering methodologies has been one of the major interests in computing science.

The nature of problems in software engineering has been addressed by Brooks (1975), McDermid (1991), and Wang et al. (1998b). A summary of fundamental characteristics of software engineering is listed below:

- Inherent complexity and diversity
- Difficulty of establishing and stabilizing requirements
- Changeability or malleability of software
- Abstraction and intangibility of software products

- Requirement of varying problem domain knowledge
- Nondeterministic and polysolvability in design
- Polyglotics and polymorphism in implementation
- Dependability of interactions between software, hardware, and human being

Along with the research and practices of software engineering, and the speedy development of the software industry, the definition of software engineering has further evolved. McDermid (1991) provided an extended definition of software engineering as follows:

> Software engineering is the science and art of specifying, designing, implementing and evolving – with economy, timeliness and elegance – programs, documentation and operating procedures whereby computers can be made useful to man.

This is representative of the second-generation definitions of software engineering.

Comparing the two definitions of software engineering, it can be seen that the former perceived software engineering as a method for software development while the latter implied that software engineering is both science and art for programming. Bearing in mind that the intention is to better represent trends and to recognize software engineering as an engineering discipline based on computer science while deemphasizing the uncontrollable and unrepeatable aspects of programming as an art, the authors [Wang et al., 1998b] offer a definition of software engineering as follows:

Definition 1.1 Software engineering is a discipline that adopts engineering approaches such as established methodologies, processes, tools, standards, organization methods, management methods, quality assurance systems, and the like to develop large-scale software with high productivity, low cost, controllable quality, and measurable development schedules.

In order to analyze the differences between the three generations of definition, a comparison of the implications and extensions of them is listed in Table 1.1. The table shows how the understanding of software engineering can be greatly improved by contrasting the perceived nature of software engineering as well as its means, aims, and attributes.

Table 1.1
Analysis of Representative Definitions of Software Engineering

No.	Nature	Means	Aims	Attributes of Aims
1	A method	Generic engineering Principles	Software	- economy - reliability - efficiency
2	The science and art	Life cycle methods: - specification - design - implementation - evolving	Program and document	- economy - timeliness - elegance
3	An engineering discipline	Engineering approaches: - methodologies - processes - tools - standards - organizational methods - management methods - quality assurance systems	Large scale software	- productivity - quality - cost - time

It is noteworthy that the perceived nature, means, and aims together with the attributes of their definitions have developed over time. The first-generation definition proposed software engineering as a method or approach to software development; the second-generation definition focused on scientific methods and art for programming; and the third-generation definition portrays software engineering as an engineering discipline for large-scale software development in an industrialized context.

1.3 A Perspective on the Foundations of Software Engineering

Having provided an improved understanding of the implications and extensions of software engineering as in Section 1.2, this section attempts to briefly investigate the foundations of software engineering from the perspectives of philosophy, theory and mathematics, and management science.

1.3.1 PHILOSOPHICAL FOUNDATIONS OF SOFTWARE ENGINEERING

Software engineering is a unique discipline that relies on special philosophical foundations at the top level. By contrasting the nature of software engineering with other engineering disciplines, it is clear that there are a number of interesting fundamental differences between them, as described below.

1.3.1.1 Virtualization vs. Realization

Given manufacturing engineering as an exemplar of conventional engineering, the common approach moves from abstract to concrete, and the final product is the physical realization of an abstract design. However, in software engineering, the approach is reversed. It moves from concrete to abstract. The final software product is the virtualization (coding) and invisible representation of an original design that expresses a real world problem. The only tangible part of a software product is its storage media or its run-time behaviors. As illustrated in Figure 1.1, this is probably the most unique and interesting feature of software engineering.

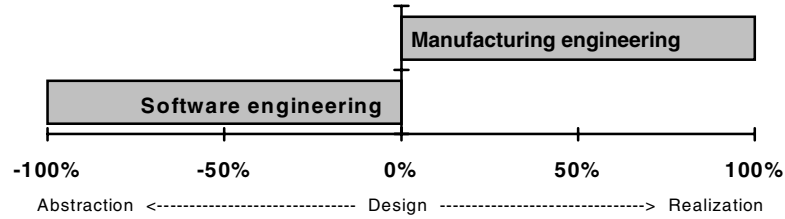


Figure 1.1 Difference between software engineering and other manufacturing engineering

1.3.1.2 Problem Domains: Infinite vs. Limited

The problem domain of software engineering encompasses almost all domains in the real world, from scientific problems and real-time control to word processing and games. It is infinitely larger when compared with the specific and limited problem domains of the other engineering disciplines. This stems from the notion of a computer as a universal machine, and is a feature fundamentally dominating the complexity in engineering implementation of large-scale software systems.

1.3.1.3 Design-Intensive vs. Repetitive-Production

As demonstrated in Figure 1.2, software development is a design-intensive process rather than a mass production process. In Figure 1.2 the design activities include specification, design, implementation, test, and maintenance; the production activities consist of duplication and package.

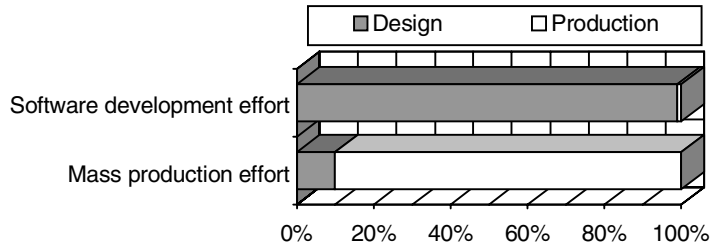


Figure 1.2 Effort distribution in software development and mass production

1.3.1.4 Process Standardization vs. Product Standardization

Directly related to the fact that software engineering is design intensive, it is recognized that the development of specific application software is characterized as mainly a one-off activity in design and production. This is because there are fewer standard software applications or products that can be mass produced save a few kinds of system software or general utilities.

Thus, for the design-intensive software development, the only element that can possibly be standardized and reused significantly are mainly the software engineering processes, not the final products themselves as in other manufacturing engineering disciplines.

1.3.1.5 Universal Logic Description vs. Domain-Specific Description

Software engineering adopts only a few fundamental logical structures, for example: sequence, condition, iteration, recursion, and concurrency. However, these provide a powerful descriptive and abstractive capability for dealing with the real-world problems.

By contrast, in other engineering disciplines, domain-and-application-specific notations have to be adopted that have limited descriptivity.

1.3.1.6 Software-Based Products vs. Physical Products

The creation as software of conventional physical products by the use of programmable and reconfigurable parts is a new and quiet industrial revolution. The 19th Century industrial revolutions were oriented on mass production by machinery and standardized process and components [Marshall, 1938]. The development of soft systems is a revolution that transforms the information processing and intelligent parts of the conventional physical products into software.

Based on the above discussion it might be argued that software engineering has become a discipline that is at the root of the knowledge structure of most engineering disciplines. The philosophical considerations explored in this subsection have attempted to clarify a set of fundamental characteristics of software engineering. These considerations also provide a basis for judging the soundness or unsoundness of specific technical solutions for software engineering, while not losing the sight of the woods for the trees.

1.3.2 THEORETICAL FOUNDATIONS OF SOFTWARE ENGINEERING

In theoretical computer science, the mathematical, logical, algebraic, and functional foundations of software engineering and programming methodologies have been studied. An outline structure of the theoretical and mathematical foundations of software engineering is described in Table 1.2.

Table 1.2
Structure of the Mathematical Foundations of Software Engineering

No.	System	Category	Branch
1	Applied mathematics		
1.1		Discrete mathematics	
1.1.1			Set theory
1.1.2			Mathematical logic
1.1.3			Functions
1.1.4			Relations
1.2		Advanced mathematics	
1.2.1			Abstract algebra
1.2.2			Process algebra
1.2.3			Category theory
1.2.4			Domain theory
1.3		Relevant mathematics	
1.3.1			Numerical methods

1.3.2			Probability theory
1.3.3			Graph theory
1.3.4			Queuing theory
1.3.5			Fuzzy logic
1.3.6			Statistics
2	Theoretical computing		
2.1		Classical theory	
2.1.1			Automata theory
2.1.2			Formal language theory
2.1.3			Computability theory
2.1.4			Algorithm complexity
2.1.5			Abstract data structures
2.2		Formal methods	
2.2.1			Algebraic specification
2.2.2			Process algebra (eg., CSP, CCS)
2.2.3			Model-oriented specification (eg., Z, VDM)
2.2.4			Refinement
2.2.5			Formal implementation
2.2.6			Verification and correction proof
2.2.7			Concurrent processing
3	Relevant theories		
3.1		Systems theory	
3.2		Information theory	
3.3		Measurement theory	
3.4		Cognitive theory	
3.5		Artificial intelligence	

In Table 1.2, the basic discrete mathematics (1.1) [Grassman and Tremblay, 1995] and classical theory of computing (2.1) [Hoare, 1969/89; Dijkstra, 1965/68/72; Knuth, 1974; Liskov and Zilles, 1974; Stoy, 1977; Gersting, 1982; Lewis and Papadimitriou, 1988; Bovet and Crescenzi, 1994] are assumed essential for software engineers. The formal methods (2.2) [Dijkstra, 1976; Gries, 1981; Hoare, 1985/95; Spivey, 1988; Dawes, 1991] are likely to influence the fundamental programming methods of the future, so that long term thinking engineers and managers will need to be aware of these topics. The relevant mathematics (1.3) [Hays, 1963; Waerden, 1969; Mathews, 1992; Grassman and Tremblay, 1995] and relevant theories (3) [Kolmogorov, 1933; Turing, 1936; MaCulloch and Pitts, 1943; Shannon, 1948; SSI, 1950, Ellis and Fred, 1962; Lindsay and Norman, 1972; Hartnett, 1977; Roberts, 1979; Kyburg, 1984; and Harvey, 1994] are optional for particular domain applications. The advanced mathematics (1.2) [Maclane, 1971; Hoare, 1985/86] are topics for long-term basic research in both software engineering and computer science.

1.3.3 MANAGERIAL FOUNDATIONS OF SOFTWARE ENGINEERING

Although the managerial foundations of software engineering have often been ignored in software engineering research and education, management sciences have, in fact, strongly influenced the formation of software engineering as a discipline. In tracing the history of software engineering, it has been found that many of the important concepts of software engineering, such as specification, requirement analysis, design, test, process, and quality, were borrowed or inspired by the methods and practices developed in management sciences and related engineering disciplines.

The managerial foundations of software engineering were cross-fertilized by the research in management science, systems theory, and quality system principles. A number of leading institutions, such as ISO TC176 and ASQ, are actively studying management theories and quality principles, as well as their application to software engineering. A brief structure of the management foundations of software engineering is summarized in Table 1.3.

Table 1.3
Structure of Managerial Foundations of Software Engineering

No.	Category	Subcategory
1	Basic theories	
1.1		Sociology
1.2		Anthropology
1.3		Semiotics
1.4		Linguistics
1.5		Psychology
2	Systems theory	
2.1		General systems theory
2.2		System design and analysis
2.3		System modeling and simulation
3	Management science	
3.1		Strategic planning
3.2		Operational theory
3.3		Decision theory
3.4		Organization methods
3.5		Management economics
4	Quality system principles	
4.1		Total quality management (TQM)
4.2		Business process reengineering
4.3		The Deming circle: Plan-Do-Check-Act (PDCA)

The fundamental theories of management are listed in Table 1.3(1). In this listing, sociology concerns organizational theory, anthropology addresses organizational culture, semiotics relates to the theories of communication and knowledge, linguistics studies language theory, and psychology concerns human behavior and learning.

As shown in Table 1.3(2), systems theory is used as a basis for management science and many other engineering disciplines. Systems theory was founded by Ludwig von Bertalanffy in the 1920s [SSI, 1950] in order to establish unified principles in both the natural and social sciences. Since then, books and articles on systems theory have proliferated [Ellis and Fred, 1962; Hall, 1967; Klir, 1972; Hartnett, 1977; Checkland and Peter, 1981]. Systems theory has provided interdisciplinary and strategic solutions that are qualitative and quantitative, organized and creative, theoretical and empirical, and pragmatic for a wide range of problems.

Management science, as shown in Table 1.3(3), is a scientific approach to solving system problems in the field of management. It includes operational theory [Fabrycky et al. 1984], decision theory [Keen and Morton, 1978; Steven, 1980], organization methods [Radnor, 1970; Kolb, 1970], strategic planning [Anthony, 1965; Khaden and Schultzki, 1983; William, 1991] and management economics [Richardson, 1966]. Management science provides management with a variety of decision aids and rules.

Three quality system principles, as listed in Table 1.3(4), have been developed during the 1970s and 1980s. The important quality management philosophies that are applicable to software engineering organization and management are TQM [Deming, 1982a/b/c; EFQM, 1993; Dunn and Richard, 1994], business process reengineering [Schein, 1961; Johansson et al., 1993; Thomas, 1994] and the Deming Circle [Deming, 1982a].

Studies of organization and management have, over time, covered methodologies for project management, project estimation, project planning, software quality assurance, configuration management, requirement/ contract management, document management, and human resource management. Table 1.4 provides a summary of the software engineering organization and management methodologies in practice.

The above review demonstrates that software engineering is a unique discipline with philosophical, mathematical, and managerial foundations based on interdisciplinary knowledge. Clearly it has borrowed from other disciplines and these discipline strands have combined to form the whole.

Table 1.4
Classification of Software Engineering
Organization and Management Methodologies

No.	Category	Typical Methods
1	Project management methods	Methods of metric-based, productivity-oriented, quality-oriented, schedule-driven, standard process models, benchmark analysis, checklist / milestones, etc.
2	Project estimation/ planning methods	Methods of KLOC metric, COCOMO model, the function-points, program evaluation and review technique (PERT), critical path method (CPM), Gantt chart, etc.
3	Software quality assurance methods	Methods of quality manual / policy, process review, process audit, peer review, inspection, defect prevention, subcontractor quality control, benchmark analysis, process tracking, etc.
4	Configuration management methods	Methods of version control, change control, version history record, software component library, reuse library, system file library, etc.
5	Requirement/contract management methods	Methods of system requirement management, software requirement management, standard contractual procedure, subcontractor management, purchasing management, etc.
6	Document management methods	Methods of document library, classification, access control, maintenance, distribution, etc.
7	Human resource management methods	Methods of position criteria, career development plan, training, experience exchange, domain knowledge development, etc.

1.4 Approaches to Software Engineering

As discussed in Section 1.4, software engineering is a discipline that has emerged from computer science and engineering and is based on inter-disciplinary theoretical and empirical methodologies. Initial approaches developed thus far have concentrated on technical aspects of software engineering such as programming methodologies, software development models, automated software engineering, and formal methods. While a cutting-edge approach, the software engineering process has been developed in the last decade for addressing the modern domain of software engineering. Each of these approaches will now be analyzed and examined.

1.4.1 PROGRAMMING METHODOLOGIES

A set of fundamental principles has been developed to cope with the complexity of problem specification and solution. Some of the important principles are abstraction, information hiding, functional decomposition, modularization, and reusability.

In tracing the history of programming methodologies, it can be seen that functional decomposition has been adopted in programming since the 1950s [McDermid, 1991]. In the 1970s the most significant progress in programming methodologies was structured programming [Hoare, 1972; Dijkstra, 1965/68/72] and abstract data types (ADTs) [Liskov and Zilles, 1974]. These methods are still useful in programming and software system design.

Since the 1980s object-oriented programming (OOP) [Stroustrup, 1986; Snyder, 1987] has been broadly adopted. Object-orientation technologies have inherited the merits of structured programming and ADTs, and have represented them in well-organized mechanisms such as encapsulation, inheritance, reusability, and polymorphism. The most powerful feature of OOP is the supporting of software reuse by inheriting code and structure information at object and system levels. It has been found that a built-in-test (BIT) method for OOP enables tests to be reused as that of code by extending the standard structures of object and object-oriented software to incorporate the BITs [Wang et al., 97c/98c/99b/d].

Along with the development of the Internet, and inspired by hardware engineering approaches, a new concept of programming, known as component-based programming, has been proposed. This approach is based on the "plug-in" and "add-on" software framework structure, and the broad availability of "commercial off-the-shelf (COTS)" software components.

1.4.2 SOFTWARE DEVELOPMENT MODELS

Programming methodologies have been mainly oriented on the conceptual and theoretical aspects of software engineering. A number of software development models have been introduced, among these are: the Waterfall [Royce 1970], Prototype [Curtis et al., 1987], Spiral [Boehm, 1988], the V [GMOD, 1992], Evolutionary [Lehman, 1985; Gilb, 1988; Gustavsson, 1989], and Incremental [Mills et al., 1980] models.

Supplementary to the above development models, a variety of detailed methods have been proposed for each phase of the development models. For instance, for just the software design phase, a number of design methods

have been in existence, typically flowcharts, data flow diagrams, Nassi-Shneiderman charts, program description languages (PDLs), entity-relationship diagrams, Yourdon methods, and Jackson system development. Of course, some of these methods can also be used in other phases of software development.

The software development model approach attempts to provide a set of guidelines for the design and implementation of software at system and module levels. However, this approach has been focused on technical aspects of software development lifecycles. Detailed descriptions and applications of existing software development models may be referred to in classical software engineering books [McDermid, 1991; Pressman, 1992; Sommerville, 1996; and Pfleeger, 1998].

1.4.3 AUTOMATED SOFTWARE ENGINEERING

The programming methodologies and software development models described above provide theoretical and technical approaches for software design and implementation. In order to support the methodologies and models, an automated software engineering approach has been sought through the adoption of computer and system software as supporting tools.

The applications of artificial intelligence and knowledge-based techniques in software development have been a key focus in this approach. Two categories of software engineering tools, computer-aided software engineering (CASE) and the unified modeling language (UML) tools, have been built for automatic implementation of different software development phases. A review of a variety of software engineering support tools is listed in Table 1.5.

In recent years, the development of the UML tool set [Rumbaugh et al., 1998] has been one of the major achievements in automated software engineering. The UML tools enable many phases in software development to be fully or largely automated, such as in the phases of system design, software design, and code generation. Although reports of industry application experiences of UML in large-scale software development are still expected, encouraging progress towards automated software engineering is being made.

The main technical difficulties in automating software development are requirement acquisition and specification, application domain knowledge representation, and implementation correctness proof. As discussed in the philosophical considerations (a) and (b) in Section 1.3.1, all of these problems need further fundamental study. This has led to attention being paid to formal methods as described in the next section.

Table 1.5
Classification of Software Engineering Supporting Tools

No.	Category	Subcategory	Tool Coverage
1	System analysis tools		Requirement analysis, acquisition, specification, prototyping, modeling, interface generation, framework generation, etc.
2	Software development tools		
2.1		Requirement analysis/specification tools	Requirement analysis, domain knowledge representation, specification, etc.
2.2		Programming tools	Compilers, debuggers, code generators, reuse support systems, object-banks, programming environment, etc.
2.3		Testing tools	Module, integration, system, acceptance, prototype, object, and interface testing, etc.
2.4		Maintenance tools	Reverse engineering, re-engineering, reuse library, static analysis, dynamic analysis, etc.
3	CASE tools		UML, ClearCase, Analysts Toolkit, Automate+, Bachman Set, Excelerator, IEW, LBMS, Maestro, Oracle CASE, Select, System architect, Top CASE, Unix SCCS, Yourdon ADT, etc.

1.4.4 FORMAL METHODS

Formal methods are a set of mathematics and logic-based methodologies and theoretical principles for software development. The logical, algebraic, and functional foundations of programming have been studied in formal methods. A number of applications of formal methods in safety-critical system design and program correctness proof have been reported [Hayes, 1987; Schneider, 1989]. The category theory developed in pure mathematics science has been found useful for establishing a unified foundation of formal methods [Hoare, 1995].

As structured programming and object-oriented programming solved many problems in software development in the 1970s and 1980s, formal methods now attempt to dig deeply into the nature of programming and to provide new solutions for rigorous and correction-provable software development. A knowledge structure of the formal approach to software engineering has been described in Table 1.2 (Part 2.2). Although our knowledge about the nature of programming has been greatly improved by the studies of formal methods, only a few of them, such as Z and SDL, have been directly applied in real-world software engineering.

Along with the fast growth of the Internet and the Internet-based programming environment in the 1990s, there has been evidence that the software engineering agenda has been driven by the industry and users. Technical innovations in everyday software engineering practices have been a major force in industrialized software engineering progress in recent years, but many new gaps have been found that require theoretical study in the overall software engineering fabric.

Table 1.6
Domain Coverage of the Approaches to Software Engineering

No.	Approach	Description	Coverage of SE Problems		
			Technique	Organization	Management
1	Programming methodologies	- functional decomposition - structural programming - object-oriented programming - component-based programming	H	L	L
2	Software development models	- life cycle model - waterfall model - spiral model - rapid prototype model - other combined models	H	M	L
3	Automated software engineering	- CASE tools - UML tool - other tools	H	L	L
4	Formal methods	- CSP - SDL - Z - Clean room - other	H	L	L
5	Software engineering processes	- CMM - Trillium - BOOTSTRAP - ISO/IEC 15504 - SEPRM - other	H	H	H

Notes: H – High, M – Medium, L – low

1.4.5 THE SOFTWARE ENGINEERING PROCESS

The software engineering facets described in Sections 1.4.1 through 1.4.4 have mainly concentrated on the aspects of software engineering as summarized in Table 1.6. Important areas of software engineering such as the organization and management infrastructures have been left untouched. Further, the systems processes by which software is created are so far

unaddressed. This draws attention to the emergence of a system process approach to software engineering.

The software engineering process approach concerns systematical, organizational, and managerial infrastructures of software engineering. It is necessary to expand the horizons of software engineering in this way because of the rapidly increasing complexity and scale demanded by software products. The need to improve software quality is also a driving force for managers.

In a view of domain coverage it is recognized that the conventional approaches, methodologies, and tools that cover individual subdomains of software engineering are inadequate. Thus, it makes sense to think in terms of an overarching set of approaches for a suitable theoretical and practical infrastructure that accommodates both new demands and improves on existing methodologies. An interesting way forward, which is capable of accommodating the full domain of modern software engineering, is that of the software engineering process.

1.5 The Process Approach to Software Engineering

The software process was originally studied as a software management method [Gilb, 1988; Humphrey, 1989], a quality assurance approach [Evans and Marciniak, 1987; ISO, 1991], or as a set of software development techniques [Curtis, 1987; Fayad, 1997a]. The reorientation of the software process to the software engineering process reflects recent trends in seeking an ideal means for organizing and describing the whole process framework of software engineering.

Two events in 1987, the development of CMM and of ISO 9000, marked the full emergence of the software engineering process as a new discipline. The software engineering process deals with foundations, modeling, establishment, assessment, improvement, and standardization of software processes. Generally, a process may be described as a set of linked activities that take an input and transform it to create an output. The software engineering process as a system is no different; it takes a software requirement as its input, while the software product is its output.

Definition 1.2 The software engineering process is a set of sequential practices that are functionally coherent and reusable for software engineering organization, implementation, and management. It is usually referred to as the software process, or simply the process.

Studies of the software process require an interdisciplinary theoretical and empirical basis. It is interesting to note that the term "software process" was inspired by management sciences [Eskiciogla and Davies, 1981; Bignell et al., 1985; Johansson et al., 1993]. Thus, the concept of a software process is more general than that of the conventional term "process" as developed in computer science where process is defined as "an execution of a subroutine [Brinch, 1973; Milenkovic, 1992]."

This section reviews the historical evolution of the process approach to software engineering, introduces the current process system models, and investigates problems identified in the process approach.

1.5.1 REVIEW OF HISTORY OF THE SOFTWARE ENGINEERING PROCESS

There are two main historical threads in tracing the emergence of the software engineering process approach. They are software engineering itself and management science. Research into the engineering processes and management principles in management sciences began in the 1960s [Simon, 1960; Schein, 1961; Ellis and Fred, 1962; Juran et al., 1962; Anthony, 1965; Richardson, 1966; Hall, 1967]. In the 1970s and 1980s, management science was well established in almost all branches as shown in Table 1.3 [Radnor, 1970; Grayson, 1973; Hartnett, 1977; Keen and Morton, 1978; Crosby, 1979; Brech, 1980; Juran and Gryna, 1980; Deming, 1982a/b/c; Khaden and Schultzki, 1983; Fabrycky et al., 1984; Leavitt, 1988]. Worthy of particular note are Crosby, Juran and Deming who developed the approach of quality conformity to requirements and specifications [Crosby, 1979; Juran, 1980; Deming, 1982a] and proposed a number of agendas that must be carried out in order to deliver total quality. These concepts have largely influenced software development processes and software quality assurance technology. In 1982, the Deming Circle, Plan-Do-Check-Act (PDCA), was proposed in management science studies [Deming, 1982a] and has drawn much interest in software process modeling and analysis. Then, a project designated ISO TC176 in 1987 to develop an international standard for quality systems [ISO 9000, 1991/93/94] that are applicable to a wide range of engineering systems including software engineering [ISO 9001, 1989/94] was implemented.

In the software engineering sector, research into the software engineering process can be traced to as early as 1965 in Weinwurm and Zagorski's work. However, interest in the software process was initiated in the 1970s after the so called "software crisis" [Naur and Randell, 1969; Baker, 1972; Brooks, 1975; Hoare, 1975]. The software process as a recognized branch of software engineering was formed in the 1980s following the work of Basili (1980), Aron (1983), Agresti (1986), Evans (1987), Boehm (1981/86/87), Gilb (1988), Humphrey (1987/88/89). These works led to the development of the capability maturity model (CMM) [Humphrey, 1987; Paulk et al., 1993a/b/c] and several other models, such as the IEEE Software Engineering Standards [IEEE, 1983] and British Standard BS 5750 [BSI, 1987] in the late 1980s. Since then the software engineering process has attracted much interest and recognition in software engineering research and practices.

1.5.2 CURRENT SOFTWARE ENGINEERING PROCESS METHODS AND MODELS

A number of software process models have been developed in the last decade such as TickIT [DTI, 1987; TickIT, 1987/92], ISO 9001 [ISO 9001, 1987/94], CMM [Paulk et al., 1993a/b/c/95a; Humphrey, 1987/88/89], BOOTSTRAP [BOOTSTRAP team, 1993], ISO/IEC 12207 [ISO/IEC 12207, 1995]; ISO/IEC TR 15504 (SPICE) [ISO/IEC 15504, 1997/98], and a number of regional and internal models [BSI, 1987; Trillium, 1992/94]. According to a recent worldwide survey [Wang et al., 1998a/99c], the ISO 9000 serial models are the most popular, followed by CMM and ISO/IEC TR 15504. Some regional, internal, and industry sector process models, such as Trillium, also share a significant part of application in the software industry. A previous survey of the distribution of the models in 1996 [Kugler and Rementeria, 1995] had shown a similar trend to that of the above distribution.

Based on the statistics and historical and theoretical significance, this book selects the four most used models for analysis. They are: CMM, ISO 9001, BOOTSTRAP, and ISO/IEC TR 15504 (SPICE), where SPICE is a synonym or the international project name of ISO/IEC TR 15504 during its development.

1.5.2.1 CMM

The SEI Capability Maturity Model (CMM) was initially developed as an assessment model for software engineering management capabilities [Humphrey, 1987/88]. As such it was expected that it would provide useful

measures of organizations bidding or tendering for software contracts. It was soon found that the concept of "process" for software engineering has more utility than that of capability assessment, and that software development organizations may use the capability model for internal process improvement. As a result of this deeper understanding, new practices in process-based software engineering have been emerging in the last decade. This is to be considered as one of the important inspirations arising from CMM and related research.

CMM modeled 18 key practice areas and 150 key practices [Paulk et al., 1993a/b/c]. These key practices and key practice areas were grouped into a 5-level process capability scale known as the initial, repeatable, defined, managed, and optimizing levels. Detailed description of CMM will be provided in Chapter 5, but what is significant is the systematic breakdown of software engineering activities, and the analytical judgement that the model allows.

1.5.2.2 ISO 9001

From another angle of management science looking at software engineering, ISO 9001 (1989/94) and ISO 9000-3 (1991) were developed within the suite of ISO 9000 international standards for quality systems [ISO 9000, 1991/93/94]. ISO 9001 (Quality Systems – Model for Quality Assurance in Design, Development, Production, Installation, and Servicing) and ISO 9000-3 (Quality Management and Quality Assurance Standards Part 3 – Guidelines for the Application of ISO 9000 to the Development, Supply, and Maintenance of Software) are important parts of ISO 9000, and are designed for software engineering.

ISO 9001 modeled a basic set of requirements for establishing and maintaining a quality management and assurance system for software engineering. It identified 20 main topic areas and 177 management issues, and categorized them into three subsystems known as management, product management, and development management. Perhaps because of its simplicity, ISO 9001 has been the most popular process model that is adopted in the software industry [Mobil Europe, 1995; Wang et al., 1998a]. Detailed description of ISO 9001 will be provided in Chapter 6, but an important characteristic of ISO 9001 is the underlying notion of a threshold standard and pass/fail criteria.

1.5.2.3 BOOTSTRAP

BOOTSTRAP [BOOTSTRAP team, 1993; Koch, 1993; Haase et al., 1994; Kuvaja et al., 1994], released in 1993, was an extension of the CMM model

that was customized to European ideas. A number of new technology processes and a flexible and precise rating method had been developed in BOOTSTRAP. In BOOTSTRAP, more technical and methodological process activities were tackled and, when an attempt was made add these process activities into the CMM capability model, a mixture of process and capability in a single dimension was produced. This was thought unhelpful and it began to be understood that there was a need to distinguish process (the software engineering activities) from capability (the measurement of the software engineering activities). The concept of a two-dimensional process model evolved from this work.

BOOTSTRAP modeled 3 process categories, 9 attributes, and 201 quality system attributes. These attributes are rated against 5 capability levels identical to that of CMM. However, intermediate process attributes were introduced to measure the differences between the defined capability levels. A main feature, then, is a more precise capability measure rounded to a quarter of a capability level. Detailed description of BOOTSTRAP will be provided in Chapter 7.

1.5.2.4 ISO/IEC TR 15504 (SPICE)

In 1992, the ISO/IEC JTC1 software engineering committee recognized a need to develop a new international standard for software process assessment and improvement [ISO/IEC 1992]. Then, after a six-year international collaborative project (SPICE) within the ISO/IEC JTC1/SC7/WG10, an ISO 15504 Technical Report suite was completed.

Inspired by BOOTSTRAP, ISO/IEC 15504 has recognized the value inherent in separating the "process" dimension from the "capability" dimension for a software engineering process model. As a result, a true two-dimensional process system model was developed for the first time. However, what is interesting is that in the ISO/IEC TR 15504 model, the activities for the process dimension and the attributes for the capability dimension at some points overlap. This means that there is still a need to further distinguish the process activities and the measurement attributes and indicators in the two dimensions.

ISO/IEC TR 15504 modeled 5 process categories, 35 processes, and 201 base practices. The processes are measured at 6 levels with 9 attributes. As a new 2-dimensional process model, the rating method of ISO/IEC 15504 is quite complicated. Detailed description of ISO/IEC 15504 will be provided in Chapter 8.

1.6 Issues in Software Engineering Process Research and Practices

As software engineering process study is at an early stage in its development, there are still debates concerning it. A number of criticisms have been raised about the process approach in general and CMM in particular. In a paper entitled "A Critical Look at Software Capability Evaluation," Bollinger and McGowan (1991) investigated the subjectivity and inaccuracy of the CMM process model and the process assessment methodology. Brodman and Johnson (1994) pointed out the need to tailor CMM for small software organizations, and the fact that there was no such mechanism available.

Further, a series of criticisms in a special column *in Communications of the ACM* was published recently entitled, "Software Development Process: the Necessary Evil?" [Fayad et al., 1997a] and "Process Assessment: Considered Wasteful" [Fayad, 1997b]. Although this column was mainly focused on CMM, it has triggered a lot of interesting discussion on both progress and problems inherent in the process approach and current process models.

1.6.1 PROBLEMS AND OPEN ISSUES IDENTIFIED

Generally, problems identified in the software engineering process debate may be traced to three root causes:

- Lack of formal description
- Chaotic interrelationships
- Deficiency of validation

This subsection describes the problems in the three categories. Solutions for these problems will be sought systematically throughout this book.

1.6.1.1 Problems in Process Modeling

Currently, it is seen as necessary to simplify the diversity of process models and to give legitimacy to modeling and analysis by creating a unified theory and integrated framework, and by introducing formal and algorithmic description.

(a) Basic requirements for process models

A variety of international, regional, and internal process models have been developed with various sizes, purposes, orientation, and structures. These models need to be summarized in terms of their methodologies, usage, and applications. More importantly, the following issues have to be tackled:

- What are the basic requirements for a process model?
- What should be essentially covered by a process model?
- What is the complete view of the software engineering process system?

These issues are investigated in Chapter 2.

(b) Classification of process models

The existing process models have been developed using various approaches such as checklists, independent models, derived models, empirical models, or descriptive models. Classifying the existing models from the viewpoints of model frameworks and methodologies is addressed in Chapter 2.

(c) Formal description of process models

Almost all existing process models are empirical-and-descriptive models. These models lack rigorous and formal description of model structure, process framework, adequacy rating scale, capability rating scale, and capability determination algorithm. Problems of ambiguity, instability, too much subjectivity, and inaccuracy in process assessment and application were identified in existing process models.

In Chapters 5 - 9, the frameworks of the major current models will be formally described. Their capability determination methods will be systematically elicited in order to create the algorithms for the current models. The formal approach will be demonstrated as being particularly useful for process designers, analysts, users, assessors, and tool developers, and will enable them to understand the current process system models.

1.6.1.2 Problems in Process Analysis

In analyzing current process models, one cannot avoid problems of different orientation, incompatibility, and nontransformability. This subsection briefly

describes the problems in process system analysis. A detailed exploration is provided in Chapters 10 through 12.

(a) Orientation

The current process models exhibit different orientations in software process modeling. Divisions between current process models cause many problems in comparative analysis and modeling. In order to integrate the existing models and to create a complete view of the software engineering process, this book will develop a new reference model approach in which the process systems of current models are treated as subsets of a super reference model – the SEPRM [Wang et al., 1996a/97a/98a/99e]. This will be explored in Chapter 9.

(b) Compatibility

Ensuring system compatibility is a proven successful practice in the software and computer industry. However, the compatibility of the current process models and their assessment results are found to be quite limited. By treating the current process models as subsets of SEPRM in Chapters 10 – 12, the compatibility problem can be solved without the cost of changing the existing models.

(c) Transformability

Comparability and transformability between models are fundamental requirements for a mature scientific discipline. By relating the assessment results of process capability levels between the current process models, a software development organization can avoid being assessed several times as required by different process models at very high cost. However, transformability between the current process models has never been studied and it seems quite a hard problem for the conventional one-to-one mapping approach. By treating current models as subsets of SEPRM and by establishing quantitative transformability, the capability transformation problem will be solved in Chapter 12.

1.6.1.3 Problems in Model Validation

Current process models have only provided some informal discussions on rationales at the middle (process) level. The validation of a process system both at the highest (organization) and lowest (attribute) levels are conspicuous by their absence. At the top level, a least-complete set of fundamental process categories of a software development organization has

to be identified and modeled. At the bottom level, the attributes of a complete set of process practices should be identified and benchmarked. These points are expanded as follows:

(a) Functional process organization in a software development organization

It is observed that some models organize processes at a number of fixed capability levels, some models group processes into different management topic areas, and some models categorize processes according to life-cycle practices. This poses the question: what is the best approach for modeling the software process system at the highest level? A structure of process systems with the organization, development, and management subsystems will be investigated in Chapter 2.

(b) Benchmark of process attributes

Quantitative analysis and benchmark of process attributes are other foundations needed to validate a model at the lowest level. Reports of benchmarks for the current models are rarely to be found in literature. Therefore, a series of worldwide surveys on a superset of processes has been conducted [Wang et al., 1998a/99c]. A set of benchmarks on process attributes, such as mean weighted importance and ratios of significance, practice, and effectiveness, have been derived to validate the SEPRM reference model, and to support the analysis of the current models. The benchmarks and their applications will be addressed in Chapter 10.

1.6.2 METHODS AND APPROACHES OF THIS WORK

There should be a systematic solution to the class of the problems identified in Section 1.6.1, and this would enable a unified software engineering process system framework. To achieve this it is necessary to adopt a new set of approaches, as shown in Table 1.7, which includes a comparison with conventional methods.

The rationales of the methods and approaches adopted or developed in this book are described below.

Table 1.7
Methods and Approaches to Software Engineering Process Modeling

Aspect	Methods Used in Existing Work	Methods Developed in This Book
Modeling	- empirical-and-descriptive modeling - natural language description	- formal-and-descriptive modeling - a formal and rigorous approach
Analysis	- one-to-one - qualitative - unidirectional mapping	- many-to-many - quantitative - bidirectional mapping
Model Coverage	- individual aspects and orientation - varying overlaps	- unified fundamental framework - overarching superset model
Model Validation	- post-industry trials	- studies of theoretical foundations - characterize BPAs by quantitative attribute benchmarking - pre-industry survey of practical foundations of practices and attributes

1.6.2.1 Methods in Process System Modeling

Potential modeling techniques for software engineering process systems can be empirical/formal, descriptive/prescriptive, and qualitative/quantitative. The current process system models such as ISO/IEC TR 15504, CMM, ISO 9001, and BOOTSTRAP are empirical-and-descriptive models. These models use natural language to describe the process system models, which creates redundancy, ambiguity, and difficulty in quantification. Instead of the descriptive "what to do" for a process model, a prescriptive approach may be taken to model "how to do" in a process system. However, this is not the main goal of system modeling because it diverges from the abstraction principles used in modeling a complicated process system.

A formal and algorithmic approach is adopted in this book. The rigorous approach is suitable for describing the methodologies of existing and new process system models because it offers less ambiguity, increases accuracy, and enables quantification and tool support.

1.6.2.2 Methods in Process System Analysis

Conventional analysis methods for the process systems are mainly one-to-one, qualitative, and unidirectional. The one-to-one approach is difficult to use in exploring the whole picture of the major process system models, and the complexity in pairwise analysis of n models is found to be of an explosive exponential order. The qualitative approach is carried out at high levels of a process system, which is quite subjective and sometimes leads to contradiction between different authors. The single directional mapping from one model to another describes only one side of a coin, because the

mapping between models has been recognized as being asymmetric [Wang et al., 1997a/99e].

These are the reasons why a many-to-many, quantitative, and bidirectional approach is used to enable efficient, less subjective, and complete mapping and analysis of current process system models.

1.6.2.3 Methods for Process Model Integration

Existing process system models cover different areas of software engineering activities with varying orientation and overlaps. Generally, these models focus on different aspects of an entire software engineering process system domain. To incorporate the current process system models as member subsystems, a unified process system framework is needed. To achieve this, an overarching software process system reference model is developed in Chapter 9.

1.6.2.4 Methods for Process Model Validation

The validation of process models may be pre or postrelease. The postrelease option is less satisfactory. This is because lessons learned can only be internalized by changes to the model which, being late in the development cycle, will be costly. It is far better to validate the model prerelease, and to do this it is necessary to base the validation on quantitative methods and on large-scale surveys of the effectiveness of existing models.

1.7 Summary

In this chapter we have seen that the process approach to software engineering is a significant trend that has been recognized by both academics and the software industry.

Software engineering has been defined as a discipline that adopts engineering approaches to develop large-scale software with high productivity, low cost, controllable quality, and measurable development schedules.

Engineering approaches to large-scale software development have been identified as established methodologies, processes, tools, standards, organization methods, management methods, and quality assurance systems.

The software engineering process is a set of sequential practices, which are functionally coherent and reusable, for software engineering organization, implementation, and management. It is usually referred to as the software process, or simply the process.

For the newly expanded domain of software engineering, the existing methodologies covering individual subdomains are becoming inadequate. Therefore, an overarching approach is sought for a suitable theoretical and practical infrastructure capable of accommodating the full range of modern software engineering practices and requirements. The whole domain of software engineering is potentially covered by the process-based software engineering methodology. Research into, and adoption of, the software engineering process approach may be made to encompass all the existing approaches to software engineering.

The basic knowledge structure of this chapter is as follows:

Chapter 1. Introduction

- General
  - Purposes of this chapter
    - To investigate the nature and philosophical, mathematical, and managerial foundations of software engineering
    - To review existing approaches to software engineering
    - To explore the new approach of process-based software engineering and related issues in research and practices
- The nature of software engineering
  - Evolvement of definitions of software engineering
    - A programming method
    - A scientific branch and art
    - An engineering discipline
  - Fundamental characteristics of software engineering
    - The inherent complexity and diversity
    - The difficulty of establishing and stabilizing requirements
    - The changeability or malleability of software
    - The abstraction and intangibility of software products
    - The requirement of varying problem domain knowledge
    - The nondeterministic and polysolvability in design
    - The polyglotics and polymorphism in implementation
    - The dependability of interactions between software, hardware, and human being

- A perspective on the foundations of software engineering
  - Philosophical foundations of software engineering
  - Theoretical foundations of software engineering
    - Applied mathematics
    - Theoretical computing
    - Relevant theories
  - Managerial foundations of software engineering
    - Basic theories
    - System theory
    - Management science
    - Quality system principles
- Approaches to software engineering
  - Programming methodologies
  - Software development models
  - Automated software engineering
  - Formal methods
  - The software engineering process
- The process approach to software engineering
  - History and interdisciplinary background
  - Current software process models
    - CMM
    - ISO 9001
    - BOOTSTRAP
    - ISO/IEC TR 15504
    - SEPRM
- Issues in software engineering process research and practices
  - Problems identified
    - Problems in process modeling
    - Problems in process analysis
    - Problems in model validation
  - Methods and approaches of this work
    - Methods in process system modeling
    - Methods in process system analysis
    - Methods for process model integration
    - Methods for process model validation

The above sidebar is designed for review of the subject topics and their relations developed in this chapter.

With the understanding of the structure of theoretical foundations of software engineering and the identification of the problems in the process approach, it is necessary to develop a unifying process infrastructure through Part I to Part III in this book. The unified process framework will accommodate and integrate existing process models, and solve a large proportion of the problems identified so far in software engineering process research and practices. The unified theory and process framework will be applied in Parts IV to VI to derive practical methodologies of process system establishment, assessment, and improvement for practitioners.

The main aim of this book is to advocate a systematic and rigorous approach to process-based software engineering. It is expected that by this approach, existing chaotic problems in current process models can be solved, and a unifying process approach to software engineering can be established with fully investigated foundations and integrated methodologies.

It is also expected that this approach will help process system analysts and developers to mutually compare multimodels; practitioners to better understand existing models; assessors to easier manipulate current models with less ambiguity; and tool developers to accurately implement supporting tools.

Annotated References

The term software engineering was first advocated by Bauer in 1968 [Naur and Randell, 1969]. Many good textbooks on generic software engineering have been published in the 1990s. McDermid (1991) edited an academic reference book covering almost every aspect of software engineering and related science branches. Sommerville (1996) presented a more formal approach to software engineering. Pressman (1992) wrote a popular text for practitioners. Pfleeger (1998) published an easy reading text describing the latest development with informative resources.

Software engineering methodologies have evolved over the following orientations, and all the methodologies developed in the last three decades have shown effectiveness and broad usability:

- Programming methodologies: see Dijkstra, 1968/76; Knuth, 1974; Liskov and Zilles, 1974; Hoare, 1969/72/86; and Gries, 1981.

- Software development models: see Waterfall (Royce, 1970), Prototype (Curtis et al., 1987), Spiral (Boehm, 1988), V (GMOD, 1992), Evolutionary (Lehman, 1985; Gilb, 1988; Gustavsson, 1989), and Incremental (Mills et al., 1980).

- Case tools and automated software engineering: see Boehm et al., 1986; Wasserman, 1990; Bandinelli, 1992; Barghouti and Krishnamurthy, 1993; and Rumbaugh et al., 1998.

- Formal methods: see Dijkstra, 1976; Gries, 1981; Hoare, 1985/95; Milner, 1989; Hayes, 1987; Spivey, 1988; Dawes, 1991; and Bandinelli, 1992.

- Software processes: Weinwurm and Zagorski (1965) was recognized as the first article that discussed software process. Note that the process concept for software development was initiated earlier than the term "software engineering" [Bauer, 1968, in Naur and Randell, 1969]. For further development, see Basili, 1980; Aron, 1983; Agresti, 1986; Evans, 1987; Boehm, 1986/94; Humphrey, 1987/88/89/99; Gilb, 1988. On process-based software engineering, see Barghouti and Krishnamurthy, 1993; Garg and Jazayeri, 1995; Wang et al., 1997a/99c/e.

On computer science foundations for software engineering processes, see Dijkstra, 1965/68/72/76; Weinberg, 1971; Baker, 1972; Knuth, 1974; Liskov and Zilles, 1974; Brooks, 1975/87/95; Hoare, 1969/72/75/85/86/89/95; Stoy, 1977; Boehm, 1976/81/88; Gries, 1981; Gersting, 1982; Lewis and Papadimitriou, 1988; Spivey, 1988; Dawes, 1991; Harvey, 1994; and Bovet and Crescenzi, 1994.

On mathematical foundations for software engineering processes: see Hays, 1963; Waerden, 1969; Maclane, 1971; Hoare, 1986; Mathews, 1992; Grassman and Tremblay, 1995.

On managerial science foundations for software engineering processes, see:

- Systems theory (SSI, 1950)
- Operational theory (Fabrycky et al., 1984)
- Decision theory (Keen and Morton, 1978; Steven, 1980)
- Organization methods (Radnor et al., 1970; Kolb et al., 1970)

- Strategic planning (Anthony, 1965; Khaden and Schultzki, 1983; William, 1991)
- Management economics (Richardson, 1966)
- Quality system principles (Shewhart, 1939; Juran, 1962/80/88/89; Crosby, 1979; Deming, 1982a/b/86; Imai, 1986; Buckland et al., 1991).

A variety of software process models, international, regional, and internal, have been developed in the last decade. See TickIT [DTI, 1987, TickIT, 1987/92], ISO 9001 [ISO 9001, 1989/94], CMM [Paulk et al., 1991/93a/b/c/1995a; Humphrey, 1987/88/89], Trillium [Trillium, 1992/94], BOOTSTRAP [BOOTSTRAP team, 1993], ISO 12207 [ISO 12207, 1995], ISO/IEC TR 15504 (SPICE) [ISO/IEC 15504, 1992/93/96/97/98], and SEPRM [Wang et al., 1996a/97a/98a/99c/e].

Questions and Problems

1.1 What is the nature of software engineering? Is software engineering unique or special in relation to the other engineering disciplines?

1.2 Software engineering is dependent on interdisciplinary foundations. Can you identify any of these disciplines that software engineering is based on?

1.3 There is an argument that programming has no scientific foundations because both professionals and amateurs can write programs. Do you agree with this observation? Why?

1.4 What are the advantages of adopting a formal and algorithmic approach to process system modeling and description?

1.5 What are the advantages of adopting a formal and quantitative approach to process system analysis?