SOFTWARE ENGINEERING

Volume 1: The Development Process

THIRD EDITION

Original papers by

Antonia Bertolino • Mark J. Christensen • Jane Cleland-Huang • Hassan Gomaa
Claire Lohr • Eda Marchetti • Thomas M. Pigoski • Anne Pons
Roger S. Pressman • Robert J. Remington • Pete Sawyer • Geri Schneider
Jed Scully • Guy Tremblay • Robert L. Vienneau • Jason P. Winters

EDITED BY
Richard H. Thayer and Mark J. Christensen

FOREWORD BY
Carl K. Chang

Software Engineering

Volume 1: The Development Process

THIRD EDITION

Edited by

Richard H. Thayer and Mark J. Christensen

IEEE

IEEE COMPUTER SOCIETY

WILEY-INTERSCIENCE

A JOHN WILEY & SONS, INC., PUBLICATION

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

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

ISBN-13 978-0-471-68417-6
ISBN-10 0-471-68417-1

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

Software Engineering

Volume 1: The Development Process

THIRD EDITION

TAKING the EXAM

with

Foreword

This is the first of two volumes published by the IEEE Computer Society to assist the reader in preparing to take the examination required to become a Certified Software Development Professional (CSDP).

The Computer Society has taken a two-fold approach to help developers quickly and economically produce software that meets users' needs through certification and standards. These approaches are closely related, because disciplined development of high-quality software requires a good development process, standards are a key part of the process, and certification recognizes the importance of a standards-supported process.

IEEE/EIA-STD-12207 provides a framework, both business and technical, within which all the processes, activities, and tasks of software development take place. Standard 12207 partitions all the processes of software development into two classes: primary and supporting. Accordingly, the two volumes of this study guide discuss these two classes separately. Volume 1 focuses on the primary development, processes, whereas Volume 2 focuses on the supporting processes.

The primary processes discussed in Volume 1 are immediately recognizable as those commonly associated with specifying, designing, implementing, testing, and maintaining software. In addition to the specific discussions of those topics, Volume 1 includes papers that address the general issues of software engineering, including the societal context in which software developers work. Volume 1 includes discussions of:

- The key concepts and activities of software and systems engineering
- The societal and legal contexts within which software development occurs
- The key IEEE Software Engineering Standards
- The importance of software requirements and methods for developing them
- Key concepts and methods of software design
- Guidelines on the selection and use of tools and methods
- The major issues and activities of software construction
- The types of software testing that occur during development and some methods for performing software tests
- Preparing for and executing a program of software maintenance

Many of the papers in this volume examine the knowledge areas identified in the Software Engineering Body of Knowledge (SWEBOK) as being critical to the successful development of software products. In many cases, explicit cross referencing to the SWEBOK is provided.

Volume 2 focuses on the supporting processes of configuration management, validation and verification, quality assurance, reviews, audits, and documentation, together with the processes of the organizations involved in software development, including management, infrastructure, the education and training of staff, and the implementation of a program of systematic improvement of all of the software life-cycle processes.

Volumes 1 and 2 of this Third Edition revise and update the 2002 tutorial of the same title. They consist of both original papers and authoritative existing papers from IEEE archival journals and other well-regarded sources. In keeping with the importance of good standards in the development processes, excerpts from appropriate Computer Society software engineering standards have been included in both volumes.

The two volumes of the study guide have been written to aid the reader both in achieving an understanding of standards-based software development (particularly using IEEE-STD-12207) and in gaining the knowledge needed to pass the CSDP examination. Thus, any software professional, whether or not he or she wishes to obtain certification by the prestigious IEEE Computer Society, will benefit from the knowledge of proven standards-based development practices contained in these volumes.

As Past President of the Computer Society I encourage all software engineers to use the study guide and to make processes, standards, and certification a central part of their careers. I also strongly encourage them to prepare for and take the CSDP examination. Such study will benefit them directly in their career by exposing them to processes and concepts that they may not have encountered in their earlier education and training, as well as allowing them to raise their own professional stature by passing the examination. It will also raise the visibility and professional level of the entire profession, which will benefit all serious practitioners of software development who approach their work tasks with a professional attitude.

Carl Chang
President 2004, IEEE Computer Society

Preface

The continuing development of the disciplines that make up software development has reached a point of maturity that has allowed the community to reach fairly definitive judgments of what constitutes good engineering practice. The professional societies dedicated to the promulgation of good software engineering practices, as well as research and continuing education in the field, have undertaken multiple efforts to bring these practices forward to the diverse membership of the computing fraternity. This has been done through a variety of magazines, journals, textbooks, and benchmark publications that identify key areas of knowledge, along with standards that describe and prescribe the processes and activities used by software engineers. Finally, to recognize individuals who have striven to learn and apply these tools to improve the quality of their work and the products they deliver, the IEEE Computer Society has created the Computer Software Development Professional (CSDP) certification program. The keystone of the CSDP certificate is the successful completion of the CSDP examination, which is offered by the Computer Society (www.computer.org).

This two-volume work is intended to support the reader in their efforts to improve themselves and their products as well as, in particular, to help them prepare for the CSDP examination. The editors, together with the contributing authors and reviewers, have dedicated their efforts toward those two objectives.

There are two guiding themes that will be repeated throughout the two volumes. The first theme is the use of IEEE/EIA Standard 12207 as a reference. IEEE/EIA 12207 provides a conceptual model for the major business and technical processes that have, over time, been recognized as key to successful software development. The other guiding theme is the use of the Software Engineering Body of Knowledge (SWEBOK) as a reference for key areas of technical knowledge. The community of software professionals has, through the SWEBOK, reached consensus on the key areas with which a software development professional should be familiar. The goal of the papers in these two volumes is to expand on those key processes and knowledge areas to provide the reader with useful, practical information in their working career as well as in their preparation for the CSDP examination.

The two volumes, following the breakdown of 12207, partition the processes and knowledge areas into those normally thought of as constituting software development and those that support (or create the environment for) the development processes. Thus, the primary processes discussed in the first volume include the general technical concepts of systems and software engineering, along with requirements specification, architectural design, detailed design, code and unit test, and integration. The supporting but equally important processes of configuration management; validation and verification; quality assurance; reviews, audits, and documentation; management; infrastructure; the education and training of technical staff; and the implementation of a program of systemic improvement of all of the software lifecycle processes are discussed in the second volume.

The editors hope that these two volumes will be of use to the individual software developer who seeks to improve his or her levels of proficiency as well as professional standing by taking the CSDP examination. The editors also hope that this work furthers and encourages the collective improvement of the profession.

RICHARD THAYER
MERLIN DORFMAN
MARK CHRISTENSEN

Contributors

Dr. A. Frank Ackerman, Ferrum College, a.f.ackerman@ieee.org
Prof. Dr. Friedrich L. Bauer, Institut für Informatik der Technischen Universität München (retired), Munich, Germany
Dr. Doug Bell, Sheffield Hallam University, d.h.bell@shu.ac.uk
Prof. Keith H. Bennett, University of Durham, keith.bennett@durham.ac.uk
Dr. Antonio Bertolino, Istituto di Elaborazione della Informazione, antonio.bertolino@isti.cnr.it
Dr. Barry W. Boehm, University of Southern California, boehm@sunset.usc.edu
Prof. Pearl Brereton, University of Keele, o.p.brereton@cs.keele.ac.uk
Prof. David Budgen, University of Keele, d.budgen@cs.keele.ac.uk
Dr. Carl K. Chang, Iowa State University, c.chang@computer.org
Ms. Mary Beth Chrissis, Software Engineering Institute (SEI), mb@sei.cmu.edu
Dr. Mark J. Christensen. Independent Consultant, markchri@concentric.net
Dr. Jane Cleland- Huang, DePaul University, jhuang@cs.depaul.edu
Dr. Bill Curtis, Borland Software Corporation, curtis@acm.org
Mr. Jon K. Digerness, North Coast Graphics, j.digerness@juno.com
Dr. Merlin Dorfman, Cisco Systems, dorfman@computer.org
Dr. Richard E. Fairley, Oregon Health and Science University, dfairley@cse.ogi.edu
Dr. Stuart R. Faulk, University of Oregon, faulk@cs.oregon.edu
Prof. Alfonso Fuggetta, Poliecnico di Milano, alfonso.fuggetta@cefriel.it
Mr. Roger U. Fujii, Northrop Grumman Mission Systems, roger.fujii@ngc.com
Dr. Hassan Gomaa, George Mason University, hgomaa@gmu.edu
Prof. Patrick (Pat) A.V. Hall, The Open University, p.a.v.hall@open.ac.uk
Dr. Lingzi Jin, Cherwell Scientific Ltd., lingzi@cherwell.com
Ms. Beth Layman, Borland Software Corporation, beth.layman@borland.com
Ms. Claire L. Lohr, Lohr Systems, lohrsys@erols.com
Ms. Eda Marchetti, Istituto di Elaborazione della Informazione, eda.marchetti@isti.cnr.it
Mr. John J. Marciniak, Visiting Scientist (SEI), jmarcin222@aol.com
Dr. Ian Morrey, Sheffield Hallam University, i.c.morrey@shu.ac.uk
Ms. Linda M. Northrop, Software Engineering Institute (SEI), lmn@sei.cmu.edu
Dr. Ronald E. Nusenoff, Cisco Systems, nusenoff@cisco.com
Dr. James D. Palmer, Consultant, jdpalmer@ix.netcom.com
Dr. Mark Paulk, Carnegie Mellon University, mcp@cs.cmu.edu
Dr. Anne Pons, Univérsity du Québec à Montréal, pons.anne@uqam.ca
Dr. Roger S. Pressman, R.S. Pressman & Associates, Inc., pressman@rspa.com
Mr. John Pugh, Carleton University, johnpugh@sympatico, ca
Dr. Robert J. Remington, Lockheed Martin Space Systems Company, bob.remington@lmco.com
Mr. Paul Rook (deceased), City University, London, United Kingdom
Dr. Pete Sawyer, Lancaster University, sawyer@comp.lancs.ac.uk
Ms. Geri Schneider, Wyyzzk, Inc., geri@txt.com
Prof. Jed Scully, McGeorge School of Law, jscully@uop.edu
Mrs. Laura Sfardini, Politecnico di Milano, laura.sfardini@cefriel.it
Prof. Ian Sommerville, Lancaster University, is@comp.lancs.ac.uk
Dr. Richard D. Stutzke, Science Applications International Corporation (SAIC), Richard.d.stutzke@saic.com
Prof. A.G. Sutcliffe, University of Manchester, a.g.sutcliffe@co.umist.ac.uk
Dr. Richard H. Thayer, CSDP, Software Management Training, r.thayer@computer.org
Mr. Steve Tockey, Construx Software, steve.tockey@construx.com
Dr. Guy Tremblay, Univérsity du Québec à Montréal, tremblay.guy@uqam.ca
Mr. Leonard L. Tripp, Consultant, l.tripp@computer.org
Mr. Robert L. Vienneau, ITT Industries, rvien@dreamscape.com
Ms. Dolores R. Wallace, National Institute of Science and Technology (retired), drwallace1@comcast.net
Mr. Charles V. Weber, Borland Software Corporation, charlie.weber@boreland.com
Mr. Jason P. Winters, Wyyzzk, Inc., jason@txt.com.

Reviewers

Dr. Danial M. Berry, University of Waterloo
Mr. Ken Chizinsky, Cisco Systems, Inc.
Dr. James M. Conrad, University of North Carolina at Charlotte
Mr. Paul R Croll, Computer Science Corporation
Dr. Richard E. Fairley, Oregon Health & Science University
Prof. Alfonso Fuggetta, Politecnico de Milano
Mr. Roger U. Fujii, Northrop Grumman Mission Systems
Mr. Mark A. Grant, Grant & Associates
Dr. Jane Cleland- Huang, DePaul University
Mr. Barry S. Johnson, Shooting Star Solutions, LLC
Dr. Xiaoping Jia, DePaul University
Dr. Steve King, University of York
Ms Susan K. Land, CSDP, Northrop Grumman Information Technology
Ms. Claire L. Lohr, Lohr Systems
Mr. John Marciniak, Visiting Scientest - SEI
Mr. James W. Moore, CSDP, The MITRE Corporation
Prof. Mauro Pezzè, Politecnico di Milano, Bicocca
Dr. Jesse Poore, University of Tennessee
Dr. Jim Isaak, Southern New Hamshire University
Dr. Stephen Seidman, New Jersey Institute of Technology
Mr. Matthew Sheranko, Knowledge Code, Inc.
Dr. Richard D. Stutzke, Science Applications International Corporation
Mr. Steve Tockey, Construx Software
Dr. Charles M. Waespy, Institute of Defense Analysis

Contents

Chapter 1

Software Engineering Development Process

1. INTRODUCTION

This chapter discusses what software engineering is, how it arose, the nature of its relationship to the broader issues of systems engineering, and what skills and knowledge practitioners should possess in order that they may perform their tasks in a professional manner. The goal of this chapter is not to replicate or contradict the specifics of any particular topic covered in later chapters of this volume or those of the companion volume. Rather, this chapter, together with Chapter 2, is intended to frame the broader issues within which the specific processes, activities, and tasks of the software development professional are performed.

2. RELATIONSHIP TO THE SWEBOK AND SOFTWARE ENGINEERING STANDARDS

The Software Engineering Body of Knowledge (SWEBOK) project was undertaken by the IEEE Computer Society in an effort to identify, document, and describe the key areas of knowledge that were, by a consensus vote of members of the profession, necessary for the proper development of software products. In 2004 the SWEBOK, already 4 years old, was revised to encompass new and changed areas of knowledge, and to reflect the experiences of the profession in applying the earlier versions. Of necessity, such an important document is a formal one and is ill-suited to direct application to pedagogic or training efforts. Similarly, the IEEE has been developing a series of Standards, intended to identify and foster the rigorous development of software products. These two efforts of the community do not stand in contradiction to one another. Rather, they are mutually supportive, with the SWEBOK identifying areas of knowledge that software practitioners should be familiar with, while the Standards provided a concrete identification of what processes, activities, and tasks are to be performed, together with document formats. Thus the SWEBOK identifies the knowledge, while the Standards identify processes, activities, and tasks along with specific guidance, templates, checklists, and formats needed to accomplish those objectives. Neither is tutorial or discursive in nature.

The purpose of this Chapter of this volume is to provide the reader with a broader context in which those two documents, and later parts of this and the companion volume, may be contemplated and acted upon.

3. THE PAPERS

This chapter contains four papers that discuss the general processes of software engineering, forming the general background and context for the balance of the two volumes:

Software Engineering by Roger S. Pressman, which provides an overview of the tasks that software engineers must perform. Dr. Pressman has graciously revised and updated his important earlier paper on this subject specifically for this volume. Reading this paper provides an excellent background on the goals, issues, and challenges faced by practitioners attempting to properly engineer software products. The reader will find this overview paper instructive when referencing virtually any of the IEEE Software Standards and likewise serves as a useful introduction to the entire SWEBOK.

The Origin of Software Engineering by Friedrich L. Bauer, which describes the historical background behind the term *software engineering* in a highly personal and direct manner. As the reader digests this paper it will be instructive to reflect on both how much and how little has changed since the 1960's, when the events chronicled in this note occurred. Many of the issues discussed in the paper remain with us today. While not tied directly to any of the specific IEEE Standard or chapter of the SWEBOK, this paper remains instructive as it reminds us of the fundamental nature of the problems that must be overcome as the software engineering profession matures.

Software Systems Engineering by Richard H. Thayer, which frames the role of software engineering within the broader framework of systems engineering; the design, construction, and test of systems with human, hardware (computational and otherwise), and software components. As the reader works through the subsequent chapters of both volumes of this tutorial they will find it useful to constantly remember that *software* development usually occurs within the broader context of *system* development. The reader will find this paper instructive when referencing IEEE Standard 1220, *Systems Engineering Process* and IEEE Standard 1362, *Concept of Operations.*

The most relevant areas of SWEBOK are Chapter 8, *Software Engineering Management,* Chapter 9, *Software Engineering Process,* and Chapter 12, *Related Disciplines.*

Recommended Skills and Knowledge for Software Engineers by Mr. Steve Tockey, which presents a summary discussion of both the capabilities (skills to perform tasks) and knowledge (the background needed to acquire those skills) needed by modern software engineers. Reading this paper in conjunction with the SWEBOK will provide the reader with insight into the dual nature of software engineering knowledge: On the one hand, many of the principles that we must follow and apply are long lasting. On the other hand, the specifics of how they are implemented are highly subject to change. As the reader works through this paper they may find it useful to note which of the various skill and knowledge areas they have mastered and the relative importance of those capabilities in their day-to-day work. It is also a useful exercise to compare the views of the SWEBOK with those of Mr. Tockey.

Software Engineering

Roger S. Pressman, Ph.D.

As software engineering moves into its fourth decade, it suffers from many of the strengths and some of the frailties that are experienced by humans of the same age. The innocence and enthusiasm of its early years have been replaced by more reasonable expectations (and even a healthy cynicism) fostered by years of experience. Software engineering approaches its midlife with many accomplishments already achieved, but with significant work yet to do.

The intent of this paper is to provide a survey of the current state of software engineering and to suggest the likely course of the aging process. Key software engineering activities are identified, issues are presented, and future directions are considered. There will be no attempt to present an in-depth discussion of specific software engineering topics. That is the job of other papers presented in this book.

1. SOFTWARE ENGINEERING—A LAYERED TECHNOLOGY

Although hundreds of authors have developed personal definitions of software engineering, a definition proposed by Fritz Bauer [2] at the seminal conference on the subject still serves as a basis for discussion:

> [Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

Almost every reader will be tempted to add to this definition. It says little about the technical aspects of software quality; it does not directly address the need for customer satisfaction or timely product delivery; it omits mention of the importance of measurement and metrics; it does not state the importance of a mature process. And yet, Bauer's definition provides us with a baseline. What are the "sound engineering principles" that can be applied to computer software development? How do we "economically" build software so that it is "reliable?" What is required to create computer programs that work "efficiently" on not one but many different "real machines?" These are the questions that continue to challenge software engineers.

Software engineering is a layered technology. Any engineering approach (including software engineering) must rest on an organizational commitment to quality (Figure 1). Total Quality Management, Six Sigma, and similar philosophies foster a continuous process-improvement culture, and it is this culture that ultimately leads to the development of increasingly more mature approaches to software engineering. The bedrock that supports software engineering is a quality focus.

The foundation for software engineering is the process layer. Software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. *Process* defines a framework for a set of *key process areas* [3] that must be established for effective delivery of software engineering technology. The key process areas form the basis for management control of software projects, and establish the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are established, quality is ensured, and change is properly managed.

Software engineering *methods* provide the technical "how to's" for building software. Methods encompass a broad array of tasks that include: requirements engineering and analysis, design, program construction, testing, and software maintenance. Software engineering methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

Software engineering *tools* provide automated or semiautomated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called *computer aided software engineering* (CASE), is established. CASE combines software, hardware, and a software engineering database (a repository containing important information about analysis, design, program construction, and testing) to create a software engineering environment that is analogous to CAD/CAE (computer aided design/engineering) for hardware.

Portions of this paper (including the figures) were adapted from *Software Engineering: A Practitioner's Approach,*6th ed. [1] and are used with the permission of McGraw-Hill.

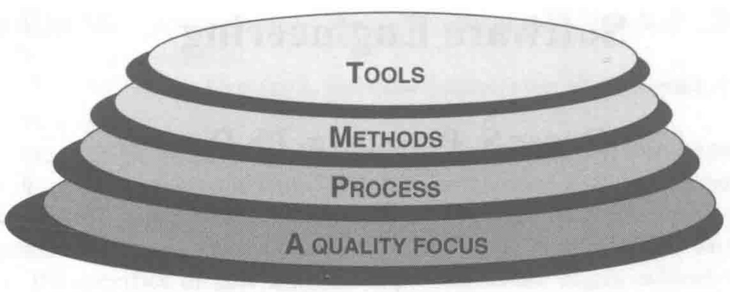

Figure 1 Software engineering layers.

2. A PROCESS FRAMEWORK

A process framework establishes the foundation for a complete software process by identifying a small number of *framework activities* that are applicable to all software projects, regardless of their size or complexity. In addition, the process framework encompasses a set of *umbrella activities* that are applicable across the entire software process.

Each framework activity (Figure 2) is populated by a set of *software engineering actions*—a collection of related tasks that produces a major software engineering work product (e.g., design is a software engineering action). Each action is populated with individual *work tasks* that accomplish some part of the work implied by the action.

The following *generic process framework* (used as a basis for the description of process models) is applicable to the vast majority of software projects:

- **Communication.** This framework activity involves heavy communication and collaboration with the customer (and other stakeholders[1]) and encompasses requirements gathering and other related activities.
- **Planning.** This activity establishes a plan for the software engineering work that follows. It describes the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule. It also addresses a team's approach to quality assurance and configuration management.
- **Modeling.** This activity encompasses the creation of models that enable the developer and the customer to better understand software requirements and the design that will achieve those requirements.
- **Construction.** This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the design or code.
- **Deployment.** The software (as a complete entity or as a partially completed increment) is delivered to the customer who evaluates the product and provides feedback based on the evaluation.

These five generic framework activities can be used during the development of small, simple programs; the creation of large Web applications; and for the engineering of large, complex computer-based systems. The details of the software process will be quite different in each case, but the framework activities remain the same.

The common process framework is complemented by a number of *umbrella activities*. Typical activities in this category include the following:

- **Software project tracking and control** allows the software team to assess progress against the project plan and take necessary action to maintain schedule.
- **Risk management** assesses risks that may affect the outcome of the project or the quality of the product.
- **Software quality assurance** defines and conducts the activities required to ensure software quality.
- **Formal technical reviews** assess software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.
- **Measurement** defines and collects process, project, and product measures that assist the team in delivering software that meets customer's needs and can be used in conjunction with all other framework and umbrella activities.

[1]A *stakeholder* is anyone who has a stake in the successful outcome of the project—business managers, end users, software engineers, support people, and so on. Rob Thomsett jokes that "a stakeholder is a person holding a large and sharp stake. . . . If you don't look after your stakeholders, you know where the stake will end up."

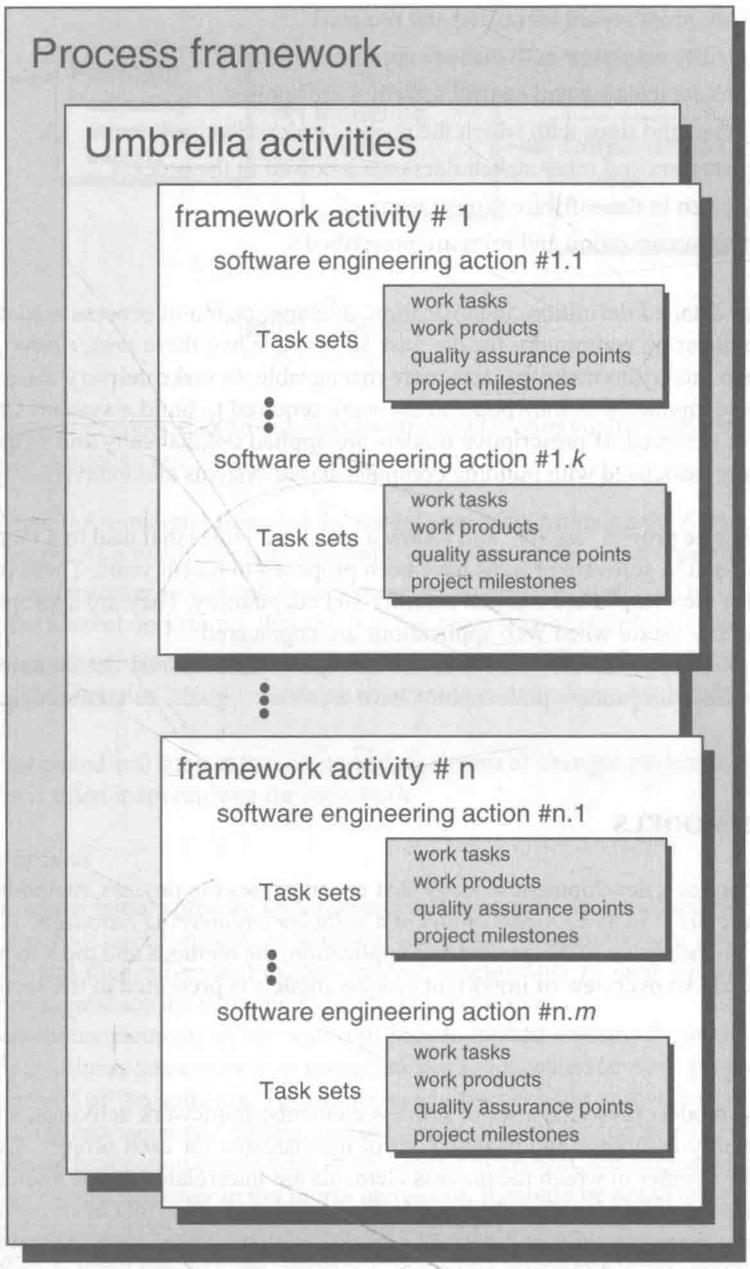

Figure 2 A software process framework.

- **Software configuration management** manages the effects of change throughout the software process.
- **Reusability management** defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.
- **Work product preparation and production** encompasses the activities required to create all work products such as models, documents, logs, forms, and lists.

All process models can be characterized within the process framework shown in Figure 2. Intelligent application of any software process model must recognize that adaptation (to the problem, to the project, to the people doing the work, and to the organizational culture) is essential for success. But process models do differ in fundamental ways:

- The overall flow of activities and tasks and the interdependencies among activities and tasks
- The degree to which work tasks are defined within each framework activity