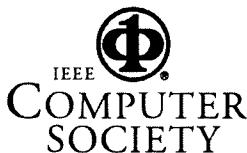


# **Software Maintenance Management**



#### Press Operating Committee

##### Chair

Roger U. Fujii,

##### Vice President

*Northrop Grumman Mission Systems*

##### Editor-in-Chief

Donald F. Shafer

##### Chief Technology Officer

*Athens Group, Inc.*

#### Board Members

**John Horch**, *Independent Consultant*

**Mark J. Christensen**, *Independent Consultant*

**Ted Lewis**, *Professor Computer Science, Naval Postgraduate School*

**Hal Berghel**, *Professor and Director, School of Computer Science, University of Nevada*

**Phillip Laplante**, *Associate Professor Software Engineering, Penn State University*

**Richard Thayer**, *Professor Emeritus, California State University, Sacramento*

**Linda Shafer**, *Professor Emeritus University of Texas at Austin*

**James Conrad**, *Associate Professor UNC- Charlotte*

**Deborah Plummer**, *Manager- Authored books*

#### IEEE Computer Society Executive Staff

**David Hennage**, *Executive Director*

**Angela Burgess**, *Publisher*

#### IEEE Computer Society Publications

The world-renowned IEEE Computer Society publishes, promotes, and distributes a wide variety of authoritative computer science and engineering texts. These books are available from most retail outlets. Visit the CS Store at <http://computer.org/cspress> for a list of products.

#### IEEE Computer Society / Wiley Partnership

The IEEE Computer Society and Wiley partnership allows the CS Press authored book program to produce a number of exciting new titles in areas of computer science and engineering with a special focus on software engineering. IEEE Computer Society members continue to receive a 15% discount on these titles when purchased through Wiley or at [wiley.com/ieeecs](http://wiley.com/ieeecs)

To submit questions about the program or send proposals please e-mail [dplummer@computer.org](mailto:dplummer@computer.org) or write to Books, IEEE Computer Society, 100662 Los Vaqueros Circle, Los Alamitos, CA 90720-1314. Telephone +1-714-821-8380.

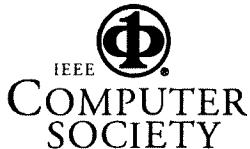
**Additional information regarding the Computer Society authored book program can also be accessed from our web site at <http://computer.org/cspress>**

# Software Maintenance Management

Evaluation and Continuous Improvement

Alain April

Alain Abran



A JOHN WILEY & SONS, INC., PUBLICATION

Copyright © 2008 by IEEE Computer Society. All rights reserved.

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

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

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

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

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print, however, may not be available in electronic formats. For more information about Wiley products, visit our web site at [www.wiley.com](http://www.wiley.com).

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

ISBN 978-0470-14707-8

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

# Contents

Foreword	xiii
<i>Thomas Pigoski</i>	
Foreword	xv
<i>Ned Chapin</i>	
Preface	xvii
<b>1 Maintenance Issues and Related Management Approaches</b>	<b>1</b>
1.1 Introduction	1
1.2 Issues in Software Maintenance	2
1.2.1 Users' Perceptions of Software Maintenance Issues	2
1.2.2 Maintainers' Perceptions of Software Maintenance Issues	4
1.3 Software Maintenance Body of Knowledge	8
1.4 Software Maintenance Definition	10
1.5 Differences Between Operations, Development, and Maintenance	10
1.6 Which Organization is Responsible for Software Maintenance?	15
1.7 Software Maintenance Standards	15
1.8 Software Maintenance Process and Activities	21
1.9 Software Maintenance Categories	23
1.10 Maintenance Measurement	23
1.10.1 Maintenance Process Measurement	23
1.10.2 Software Product Measurement	28
1.11 Service Measurement	29
1.11.1 Internal Service-Level Agreement	30
1.11.2 Maintenance Service Contracts—External Service Agreement	32
1.11.3 Outsourcing Agreements	34
1.12 Software Maintenance Benchmarking	35
1.13 Summary	37
1.14 Exercises	38
<b>2 Maturity Models in Software Engineering</b>	<b>41</b>
2.1 Introduction	41

**vi**    Contents

2.2	Overview of Basic Concepts (Process and Maturity)	42
2.3	Does CMMi Cover Software Maintenance Adequately?	45
2.4	Difference Between Maturity Model, Quality Standards, and Evaluation Method	46
2.4.1	Choosing Between ISO9001 and CMMi	46
2.4.2	The Evaluation Method	48
2.4.3	Evaluation Types	51
2.5	How is a Maturity Model Designed?	52
2.5.1	The Trillium Design Process	52
2.5.2	ISO 15504 Design Process	54
2.5.3	CMMi for Services Design Process	55
2.5.4	Summary	55
2.6	Initial Validation of a Maturity Model	55
2.6.1	IT Service CMM Model—Initial Validation Approach	56
2.6.2	CM <sup>3®</sup> Model—Initial Validation Approach	56
2.6.3	ISO 15504 Model—Initial Validation Approach	57
2.6.4	CMMI for Services Model—Initial Validation Approach	57
2.6.5	Maturity Model Validation, Conclusion	57
2.7	What is the Typical Architecture of Maturity Models?	57
2.7.1	CMMi Model Architecture	58
2.7.2	The ISO 15504 (SPICE) Model	60
2.8	An Inventory of Software Engineering Maturity Models	62
2.9	Summary	65
2.10	Exercises	66
<b>3</b>	<b>Foundations of the S3<sup>m®</sup> Process Model</b>	<b>69</b>
3.1	Introduction	69
3.2	Context of Software Maintenance	71
3.3	Proposed Classification of Software Maintenance Processes	73
3.3.1	Software Maintenance Operational Processes	75
3.3.2	Software Maintenance Support Processes	76
3.3.3	Software Maintenance Organizational Processes	76
3.4	Identification of Process Domains and Key Process Areas in Software Maintenance	76
3.5	Summary	81
3.6	Exercises	81
<b>4</b>	<b>Process Management Domain</b>	<b>83</b>
4.1	Overview	83
4.2	Maintenance Process Focus KPA	86
4.2.1	Goals of this KPA	86
4.2.2	Links with Other KPAs	86
4.2.3	Expected Results of this KPA	87
4.3	Maintenance Processes/Services Definition KPA	87
4.3.1	Goals of this KPA	87

4.3.2	Links with Other KPAs	88
4.3.3	Expected Results from this KPA	88
4.4	Maintenance Training KPA	88
4.4.1	Goals of this KPA	89
4.4.2	Links with Other KPAs	89
4.4.3	Expected Results from this KPA	89
4.5	Maintenance Process Performance KPA	90
4.5.1	Goals of this KPA	90
4.5.2	Links with Other KPAs	90
4.5.3	Expected Results from this KPA	91
4.6	Maintenance Innovation and Deployment KPA	91
4.6.1	Goals of this KPA	91
4.6.2	Links with Other KPAs	92
4.6.3	Expected Results from this KPA	92
4.7	Summary	93
4.8	Exercises	93
<b>5</b>	<b>Event/Request Management Domain</b>	<b>95</b>
5.1	Overview	95
5.2	Event/Request Management KPA	97
5.2.1	Goals of this KPA	97
5.2.2	Links with Other KPAs	98
5.2.3	Expected Results from this KPA	98
5.3	Maintenance Planning KPA	98
5.3.1	Goals of this KPA	99
5.3.2	Links with Other KPAs	99
5.3.3	Expected Results from this KPA	100
5.4	Request/Software Monitoring and Control KPA	100
5.4.1	Goals of this KPA	100
5.4.2	Links with Other KPAs	101
5.4.3	Expected Results from this KPA	101
5.5	SLA and Supplier Agreement Management KPA	101
5.5.1	Goals of this KPA	102
5.5.2	Links with Other KPAs	102
5.5.3	Expected Results from this KPA	102
5.6	Summary	103
5.7	Exercises	103
<b>6</b>	<b>Evolution Engineering Domain</b>	<b>107</b>
6.1	Overview	107
6.2	Predelivery and Transition Services KPA	109
6.2.1	Goals of this KPA	109
6.2.2	Links with Other KPAs	110
6.2.3	Expected Results from this KPA	110
6.3	Operational Support Services KPA	111

6.3.1	Goals of this KPA	111
6.3.2	Links with Other KPAs	111
6.3.3	Expected Results from this KPA	111
6.4	Software Evolution and Correction Services KPA	112
6.4.1	Goals of this KPA	112
6.4.2	Links with Other KPAs	113
6.4.3	Expected Results of this KPA	113
6.5	Verification and Validation KPA	113
6.5.1	Goals of this KPA	114
6.5.2	Links with Other KPAs	114
6.5.3	Expected Results of this KPA	114
6.6	Summary	115
6.7	Exercises	115
<b>7</b>	<b>Support for the Evolution Engineering Domain</b>	<b>117</b>
7.1	Overview	117
7.2	Configuration and Version Management KPA	120
7.2.1	Goals of this KPA	121
7.2.2	Links with Other KPAs	121
7.2.3	Expected Results of this KPA	122
7.3	Process, Service, and Software Quality Assurance KPA	122
7.3.1	Goals of this KPA	122
7.3.2	Links with Other KPAs	123
7.3.3	Expected Results of this KPA	123
7.4	Maintenance Measurement and Analysis KPA	123
7.4.1	Goals of this KPA	124
7.4.2	Links with Other KPAs	124
7.4.3	Expected Results of this KPA	124
7.5	Causal Analysis and Problem Resolution KPA	124
7.5.1	Goals of this KPA	125
7.5.2	Links with Other KPAs	125
7.5.3	Expected Results of this KPA	125
7.6	Software Rejuvenation, Migration, and Retirement KPA	125
7.6.1	Goals of this KPA	126
7.6.2	Links with Other KPAs	126
7.6.3	Expected Results of this KPA	126
7.7	Summary	127
7.8	Exercises	127
<b>8</b>	<b>Exemplary Practices—Process Management</b>	<b>129</b>
8.1	Maintenance Process Focus—Detailed Exemplary Practices	130
B.1	Level 0	130
B.1	Level 1	130
B.1	Level 2	131

8.2	Maintenance Process/Service Definition—Detailed Exemplary Practices	135
B.2	Level 0	135
B.2	Level 1	136
B.2	Level 2	136
8.3	Maintenance Training—Detailed Exemplary Practices	138
B.3	Level 0	138
B.3	Level 1	138
B.3	Level 2	139
8.4	Maintenance Process Performance—Detailed Exemplary Practices	144
B.4	Level 0	144
B.4	Level 1	144
B.4	Level 2	145
8.5	Maintenance Innovation and Deployment—Detailed Exemplary Practices	146
B.5	Level 0	146
B.5	Level 1	147
B.5	Level 2	147
<b>9</b>	<b>Exemplary Practices—Event/Request Management Domain</b>	<b>149</b>
9.1	Event/Request Management KPA—Detailed Exemplary Practices	150
C.1	Level 0	150
C.1	Level 1	150
C.1	Level 2	150
9.2	Maintenance Planning KPA—Detailed Exemplary Practices	151
C.2	Level 0	151
C.2	Level 1	151
C.2	Level 2	152
9.3	Requests/Software Monitoring and Control KPA—Detailed Exemplary Practices	159
C.3	Level 0	159
C.3	Level 1	159
C.3	Level 2	159
9.4	SLA and Supplier Agreements Management KPA—Detailed Exemplary Practices	162
C.4	Level 0	162
C.4	Level 1	162
C.4	Level 2	163
<b>10</b>	<b>Exemplary Practices—Evolution Engineering Domain</b>	<b>169</b>
10.1	Predelivery and Transition to Software Maintenance KPA—Detailed Exemplary Practices	170
D.1	Level 0	170
D.1	Level 1	170
D.1	Level 2	170

**x**    Contents

10.2	Operational Support Services KPA—Detailed Exemplary Practices	175
D.2	Level 0	175
D.2	Level 1	175
D.2	Level 2	176
10.3	Software Evolution and Correction Services KPA—Detailed Exemplary Practices	178
D.3	Level 0	178
D.3	Level 1	178
D.3	Level 2	179
10.4	Verification and Validation KPA—Detailed Exemplary Practices	182
D.4	Level 0	182
D.4	Level 1	182
D.4	Level 2	183
<b>11</b>	<b>Exemplary Practices—Support to Evolution Domain</b>	<b>187</b>
11.1	Configuration and Change Management—Detailed Exemplary Practices	188
E.1	Level 0	188
E.1	Level 1	188
E.1	Level 2	189
11.2	Process, Service, and Software Quality Assurance—Detailed Exemplary Practices	190
E.2	Level 0	191
E.2	Level 1	191
E.2	Level 2	191
11.3	Maintenance Measurement and Analysis—Detailed Exemplary Practices	193
E.3	Level 0	193
E.3	Level 1	194
E.3	Level 2	194
11.4	Causal Analysis and Problem Resolution—Detailed Exemplary Practices	195
E.4	Level 0	195
E.4	Level 1	195
E.4	Level 2	196
11.5	Software Rejuvenation, Migration, and Retirement—Detailed Exemplary Practices	197
E.5	Level 0	197
E.5	Level 1	197
E.5	Level 2	198
<b>12</b>	<b>Assessment Process, Assessment Tool, and Case Studies of the Use of S3<sup>m</sup>®</b>	<b>203</b>
12.1	Evaluation Process and Support Tools	203

12.2 Example of Evaluation Results	206
12.3 Four Case Studies Using S3 <sup>m®</sup>	210
12.3.1 Contributions to the Definition of Software Maintenance	210
12.3.2 Contributions to the Definition of the Service Level Agreement (SLA)	210
12.3.3 Contributions to Software Product Quality Assessment During Predelivery and Transition	211
12.3.4 Contributions to the Improvement of a Very Small Maintenance Function	213
12.4 Summary	221
<b>13 Summary</b>	<b>223</b>
13.1 The Maintenance Issues Revisited	224
13.2 Questions 1 and 2—Is Maintenance a Specific Domain of Software Engineering?	224
13.3 Question 3—Does the CMMi Adequately Address Software Maintenance?	225
13.4 Question 4—What Would the Architecture of a Software Maintenance Maturity Model Look Like?	225
13.5 Question 5—How Can Such a Model be Used in Practice?	225
13.6 Lessons Learned and Contributions	225
13.7 Further Reading	226
<b>Appendix A Maintenance Standards Models and Enhancement     Proposal</b>	<b>227</b>
A.1 Software Maintenance Standards	227
<b>Appendix B Term Assignment for Students</b>	<b>231</b>
<b>Appendix C Acronyms and Glossary</b>	<b>235</b>
<b>References</b>	<b>285</b>
<b>Index</b>	<b>301</b>
<b>About the Authors</b>	<b>313</b>

# Foreword

In today's demanding business world, the need for comprehensive software maintenance has never been greater. The global market has resulted in a dizzying variety of software providers, all tasked with the decades-old question: How do we accurately maintain software in a cost-effective, timely manner while staying competitive and within budget?

Again and again, the problem boils down to one issue: Many software organizations do not have any defined processes for their software maintenance activities. Over the last four decades, several process models have been suggested but none have been widely accepted.

Recently, I was introduced to Dr. April and Dr. Abran's revolutionary method of improving software maintenance: the Software Maintenance Maturity Model (S3<sup>m®</sup>).

Capability maturity models (CMMs) are not a new idea, having been around since the early 1990s. The two most well known were both developed by Carnegie Mellon University's Software Engineering Institute (SEI): the original CMM for software, and its successor, Capability Maturity Model® Integration (CMMI®). It is the latter, CMMI® Version 1.1, that forms the basis for this newly proposed comprehensive maturity model unique to software maintenance, and this is what I think will make S3<sup>m®</sup> a success.

The SEI has compiled a tremendous amount of empirical data that clearly show the benefits of process improvement, including improved productivity, cost, quality, customer satisfaction, and return on investment. As an SEI-authorized instructor and CMMI® appraiser, I have seen firsthand the results that can be achieved by organizations that enthusiastically adopt CMMI®, making me a firm believer in maturity models.

It is obvious from this book that authors Dr. April and Dr. Abran have done an exhaustive amount of research in developing their S3<sup>m®</sup>.

They have revisited 30 years' worth of publications, dissecting past proposed maturity models, beginning with the groundbreaking work of software maintenance pioneer Mira Kajko-Mattsson, and compiled a comprehensive list of best practices that span every aspect of software maintenance. Interviews with current software maintenance specialists and managers reinforced what today's maintainers need and want in a software maintenance maturity model. The resulting model includes four process domains: Software Maintenance Process Management, Software

Maintenance Request Management, Software Evolution Engineering, and Support to Software Evolution Engineering.

Those in the software maintenance field who want to benchmark their maintenance efforts should consider implementing the model contained in this book. Dr. April and Dr. Abran have laid out the best practices from around the world, creating a road map that will revolutionize the way you work.

THOMAS PIGOSKI  
*President, TECHSOFT, Inc.  
Pensacola, Florida*

# Foreword

Many a chief information officer (CIO) or chief technology officer (CTO) sees software maintenance as a lot of relatively small projects. Typically, the projects require from a few hours to a few months of personnel time but, collectively, they may keep as much as half of the programming and analyst staff occupied. Usually, the projects are to do corrective maintenance or minor enhancement maintenance on the organization's existing software-implemented systems. For the users of those systems, such software maintenance is vital in keeping software systems working as expected and needed for the organization to thrive. How such software maintenance work is managed from first-line supervisors on up is, therefore, critically important for the organization's software systems users.

This book reports best practices for doing such software maintenance management, based on the carefully examined experiences of many maintenance managers and supervisors from many parts of the world. The authors have summarized this body of international practical experience, and presented it in this book for the use of software maintenance managers and supervisors everywhere. The book is organized around the S3<sup>m®</sup> (software maintenance maturity model). The S3<sup>m®</sup> is a maturity model because it takes into account the results managers internationally have achieved from their different choices made in managing software maintenance. The higher maturity levels are seen as yielding better benefit/cost ratios for the organization, more effective use of information systems maintenance personnel, and more satisfied users of the organization's software systems.

For any manager or supervisor of software maintenance who is a reader of this book, I recommend studying the chapters in their numeric order, with a liberal use of Appendix C for help with how terms are defined (since different cultures use terms differently). I recommend specific attention to the key practices that the authors cover, while paying close attention to the contexts in which they are reported to be most effective. And readers of this book, just like people working in software generally, should be alert to not getting so wrapped up in the details that they dim their vision of the big picture.

Here is to good practical returns to you from reading about a model of best practices in managing common small software maintenance projects!

NED CHAPIN  
*Information Systems Consultant*

# Preface

Organizations that rely on revenues from developing and maintaining software must now respond to increasingly demanding customers in a new, globally competitive market. With services and products available from vendors the world over, customers are insisting that their software systems be high quality, cost as little as possible, and be accompanied by support services that challenge those of the competition. To satisfy these demands, the dynamic organization faces two challenges: It must have the ability to develop and maintain software to meet the customer's needs, and it must have access to software that supports the company's business processes. Both software perspectives (external and internal) must be reliable and well maintained. Maintaining the mission-critical software of an organization is not an easy task, and requires that a management system for software maintenance be in place. Moreover, an adequate system for software maintenance has to satisfy a number of needs (the service criteria of a company's customers and the technical criteria of the domain), as well as maximize strategic impact and optimize the cost of software maintenance activities. This requires that the organization be committed to the continuous improvement of software maintenance processes.

The international consensus of the generally accepted knowledge on the software engineering discipline has been documented in the *Guide to the Software Engineering Body of Knowledge* (ISO TR 19759) or SWEBOK. Software maintenance has been recognized as one of the key areas of knowledge required in this new profession. To meet their customers' demands, software organizations must recognize the importance of maintenance processes and the required quality of service, and software engineers must master the skill set needed to implement these processes.

A maintenance-related improvement model must be suited to the software maintainer's reality, that is, the creation of quick, failsafe, and well-documented small enhancements to production systems.

Our book reflects this new perspective in which quality and process increasingly matter in a competitive and global services environment. Here, we share years of practical experience in software maintenance improvement with the development of a practical software maintenance maturity model, or S3<sup>m</sup>®, for use in industry. Because it is a practitioner's model, software maintenance managers and software

process improvement groups will find in it effective strategies for improving software maintenance, along with numerous illustrative examples. Applying the best practices described in this maintenance model will ensure that maintenance staff are equipped to respond to the most demanding customers, feel supported by management, and are proud to be part of the maintenance team.

In addition, the book introduces many of the theoretical concepts and references needed by professionals, managers, and students to help understand the fundamentals of the identification and evaluation of software maintenance processes and of improvements to them.

$S3^m^{\circledR}$  was created *by* software maintenance managers *for* software maintenance managers. The model is a structured and integrated collection of software engineering best maintenance practices observed and documented from many sources. It also refers to many of the best practices documented in the Capability Maturity Model Integration (CMMI<sup>TM</sup>) published by the Software Engineering Institute (SEI). These practices have been tailored to adapt to the peculiarities of queue management for randomly incoming maintenance service requests, as opposed to well-planned large software development or maintenance projects managed with project management techniques. The  $S3^m^{\circledR}$  model reflects the point of view of small teams performing relatively small maintenance tasks.

This book is intended for those maintaining software and managing a large number of small maintenance requests, the duration of which is, most of the time, less than a few days or weeks of work effort. The  $S3^m^{\circledR}$  model presented here can also be used to assess software maintenance suppliers/outsourcers of small maintenance services, and later to monitor that they do indeed implement best practices for the management of software maintenance.

Readers with extensive practical experience in software maintenance management might want to skip Chapter 2, whereas readers with extensive knowledge of maturity models might want to skip Chapter 3. This book can also be used as a textbook on software maintenance management, with relevant questions at the end of each chapter.

## STRUCTURE AND ORGANIZATION OF THIS BOOK

The book is organized into 13 chapters. Chapter 1 discusses the basics of software maintenance, that is, the set of concepts needed by the software engineer in charge of software maintenance management. This chapter includes an inventory of the software maintenance processes and activities documented in the literature. The terminology, definitions, and concepts used in the book are also introduced. Numerous references are provided and can be used to learn more about the many software maintenance processes and perspectives.

Software maturity models have gained broad market awareness and market recognition. Chapter 2 focuses on process improvement and identifies models that can help maintainers. An inventory of these models is presented and we highlight the ones that matter to the software maintainer. We also present the rationale for ini-

tiating a software maintenance improvement program. The terminology of process improvement and assessment methods is explained in detail.

Chapter 3 presents an overview of our S3<sup>m®</sup>. This overview includes a presentation of its architecture and how it was developed, as well as the high-level concepts of the model, its common five-level structure, its scope, and its objectives. We also introduce the four software maintenance “process domains” that will be introduced in Chapters 4 to 8. The detailed practices of each of these “process domains” will be presented in Chapters 8 to 11.

In Chapter 4, the software maintenance best practices have been grouped together in the process management domain. Five key process areas are presented (process focus, process/service definition, training, process performance, and innovation/deployment). Each key process area is presented and the detailed practices explained. Software maintenance training practices are closely linked to the work of Mira Kajko-Mattson.

In Chapter 5, the second set of software maintenance best practices groups together all the practices that pertain to the management, planning, and control of problem reports and change requests. Four key process areas are presented: event/service requests, planning, monitoring and control of event/service requests, and service level agreements (SLAs) and supplier agreements.

The third set of maintenance best practices presented in Chapter 6 groups together four key process areas of operational processes, covering the daily operational practices shared by software maintenance staff and software development staff, as well as software transition, operational support, software evolution/correction, and software verification/validation.

The fourth set of maintenance best practices presented in Chapter 7 groups together all the supporting processes that are often shared by maintainers and developers. It contains five key process areas: configuration management, process/product quality assurance, measurement/analysis, causal analysis and problem resolution, and software rejuvenation. Rejuvenation addresses redocumentation, restructuring, reverse engineering, and reengineering. This last key process area also includes the migration and retirement of software.

Chapters 8 to 11 present exemplary practices at levels 0, 1, and 2 of the four process domains of the S3<sup>m®</sup> model. Four case studies of the use of S3<sup>m®</sup> in industry are presented in Chapter 12 to illustrate various uses in industrial contexts. We also introduce some assessment support tools and a knowledge-based system that helps in understanding and using the maturity model.

Chapter 13 presents a brief summary of the questions addressed in this book, key findings, and lessons learned.

This book also contains three Appendices. Appendix A presents a software maintenance standards model as well as proposed enhancements to existing standards. Appendix B proposes a number of term assignments for students taking a semester course on software maintenance. Finally, Appendix C presents a list of acronyms and a glossary of terms.

Additional material to complement the information contained in this book can be found by following the links on our website at [www.s3m.ca](http://www.s3m.ca).

This book describes the basic practices of the maturity model (levels 0, 1 and 2). The advanced practices (levels 3, 4, and 5) are made available to organizations that contribute to the research and evolution of the maturity model.

This maturity model is protected by copyright and trademarks. Only consultants who have received an accreditation can use this model commercially in their organization and will receive the support of the authors. Communicate with us or check our website to ensure the validity of their accreditation.

## **ACKNOWLEDGMENTS**

This book is dedicated to all those maintenance programmers out there who are overwhelmed with urgent maintenance requests and who want to move forward to improve the valuable services they provide to their customers. We thank Nazeem Abdullatif, Ali Al-Jalahma, Dhiya Al-Shurougi, M. Arpino, Ganesh Ayyer, René Bertand, Denis Bistodeau, Jacques Bouman, Rhee Byung-Do, Luc Chaput, Gilles Chauvin, Robert Clark, Francois Coallier, Andy Crowhurst, M. De Sousa, Sujay Deb, David Déry, Alex Dobkowi, Claude Dupont, Mario Ferreira, John Flint, Bernard Fournier, J. Gagnon, Garth George, John Gartin, Mohd Ali Ghuloom, Pierre Giroux, Paul Goodwin, S. Grenier, Nic Grobbelar, Gerrie Haasbroek, Jane Huffman Hayes, Taha Hussain, Ravi Kalyanasundaram, Peter Keeys, Normand Lachance, Bruno Laguë, André Larivière, Orma Lytle, Jean Mayrand, Mac McNeil, Ettore Merlo, Hélène Michaud, Caroline Milam, Luis Molinié, James W. Moore, Sophie Mourtardier, Vanrerlei Vilhanova Ortêncio, Al Pecz, Bruno Potvin, R.Purcell, Ian Robinson, Humberto Roca, Talib Salman, John Shaughnessy, Dave Stapley, Don Stewart, Phil Trice, Frank Trifiro, Laxman N. Vasan, Roxanne White, and Mohammed Zitouni. Special thanks to David-Alexandre Paquette for the 2006 case study.

ALAIN APRIL  
ALAIN ABRAN

*Montreal, Quebec, Canada  
January 2008*

# Chapter 1

---

## Maintenance Issues and Related Management Approaches

### This chapter covers:

- Users' and maintainers' perceptions of software maintenance
- Maintenance body of knowledge and maintenance definitions
- Organization responsible for software maintenance
- Software maintenance standards
- Maintenance process and activities
- Maintenance measurement, service measurement, and benchmarking

### 1.1 INTRODUCTION

When successfully completed, the software development process ends with the delivery of a software product that should meet the customer's original requirements. Once in operation, a software product either changes or evolves as defects are uncovered, operating environments change, and new user requirements surface.

This chapter introduces the basic concepts that govern software maintenance. For instance, once the software is in operation, failures occur. Failures are defined as defects that have found their way into production without being caught by the developer's testing effort. Failures are detected and reported by users and are generally repaired under the initial software warranty or through postimplementation delivery maintenance activities. In addition to the correction of the software after its original implementation, there will also be changes to be considered as users formulate new requests. Although it is generally believed that the software maintenance cycle starts on the first day of operation of the new software, it is well documented

that many software predelivery activities go on behind the scenes during the development process itself. These and many other activities are part of the software maintainer's knowledge, and are the maintainer's responsibility.

## **1.2 ISSUES IN SOFTWARE MAINTENANCE**

A number of authors have identified maintenance-related issues that affect maintenance resource management and processes and their specific tools/techniques. The issues reported in the literature vary according to the point of view of the authors who identified and described them. These issues can be categorized based on two viewpoints:

1. External: perceptions of customers (i.e., users and stakeholders)
2. Internal: perceptions of software maintenance engineers and managers

From the customers' (external) point of view, the main issues reported are high cost, slow delivery of services, and fuzziness of prioritization [internal information systems and information technology (IS/IT) priorities versus users' priorities] [Pigoski 1997]. From the maintainers' (internal) point of view, key maintenance issues include software that has been poorly designed and programmed, and a major lack of software documentation [Glass 1992]. In addition, it has been often observed that *the biggest problem in software maintenance is not technical but rather its management*. Let us examine both viewpoints in more detail.

### **1.2.1 Users' Perceptions of Software Maintenance Issues**

It has been reported that a large portion of software life cycle costs—50 to 90%—come from software maintenance, and Boehm states that, for every dollar spent on development, two will need to be spent on maintenance [Boehm 1987].

It has also been reported that a significant part of the user's perception of the high cost of maintenance may stem from inadequate communication by maintenance managers about the type of maintenance work performed on the customer's behalf. Maintenance managers too often group enhancement and corrective work together in their management reports, statistics, and budgets, which warps costs in both maintenance budgets and reports. This aggregation maintains customer misconceptions about maintenance services and perpetuates the misleading perception that software maintenance is mainly concerned with correcting defective software.

Software maintenance has some peculiarities when compared to maintenance in other domains: whereas hardware and mechanical components deteriorate when there is no maintenance, software, by contrast, deteriorates as a result of multiple maintenance activities [Glass 1992]. Because software practices have not matured to the extent that hardware practices have, maintenance is performed by identifying

one or many components and fixing them. Software maintenance often requires re-work of many, if not all, parts of the software. Multiple maintenance activities are blamed for further degrading the software's internal structure and making future maintenance activities progressively more difficult. If no strategy is put in place to control these effects, the software structure and quality continue to deteriorate. Software in this state is often referred to as legacy software.

Software maintenance is also labor intensive, with the majority of costs arising from programmers' salaries. Indeed, with hardware costs no longer representing the biggest portion of IS/IT costs, that of hiring expert programmers now ranks first (e.g., more than \$1500 per day for the consulting services of a programmer specializing in ERP software language).

By contrast, users' perceptions of the costs are also influenced by service annoyances, which can sometimes have a considerable impact on user perceptions about both the quality of services and their cost. For example, production outages and errors in software-supported applications, as well as delays and high costs in implementing minor functional changes are often cited as problem areas in software maintenance.

Furthermore, for software under maintenance production system failures occur randomly, and user requests come in on an irregular basis. Without agreed-upon and mature queue-management mechanisms supported by detailed service-level agreements (SLAs), users will often get service that does not meet their real and stated priorities. When they get poor maintenance services, some users overreact and rank all requests as high priority and demand that all problems and requests be addressed at the same time. Given that production-system failures are random events, and that they need to be addressed first, such users perceive that work on their requests is not progressing as they would expect. When customers become frustrated with the slow delivery of services, some will consider developing local solutions to solve their problems, or might consider subcontracting or outsourcing maintenance work altogether.

Some of the users' perceptions of the slowness of service is partially based on a misunderstanding of the fundamental difference between the electronic/computer hardware maintenance and software maintenance domains. For instance, a software user often does not have a clear idea of what constitutes a software maintenance activity, describing, for example, how the repair of a broken printer or computer component may simply involve a modular-component replacement activity. He or she also might note that electronic equipment design, being more mature than software design, is based on a thorough "modularization" of components, which does not create side-effects during maintenance activities. Moreover, when well designed, hardware allows complete testing and diagnostics of components at the touch of a button! Many users, unfamiliar with the software domain, may perceive that software is maintained in a similar fashion. Unfortunately, software has not yet matured to this stage. There remain many challenges to the existence of modularity and autotesting in software, in particular because of its lack of maturity; furthermore, most software currently in use is more than a decade old and would not incorporate this type of architectural technology.

Is this user perception representative of the actual maintainers' workload? Are the maintainers spending most of their time correcting software bugs, or providing value-added services?

As far back as 1980, Lientz and Swanson [1980] were the first to point out that 55% of requests routed to maintenance organizations concerned new functions rather than failure correction. The proposed S3™® model addresses failure corrections and new functions as well as many other maintenance-related activities. This was confirmed ten years later in another study, based on data collected over a few years and for each maintenance activity category [Abran 1991]. Similarly, Pressman [2004] also notes, based on feedback collected from maintenance engineers, that, rather than spending from 50% to 80% of their effort on correcting problems, they spend the biggest portion of their effort on software evolution, on adding user-requested functionality, and on answering all kinds of questions about the business rules of application software. Unfortunately, such value-added activities are poorly communicated and rarely reported in detail to the clients as monthly maintenance accomplishments. In summary, the users' perceptions do not necessarily map well to the actual maintainers' workload: "The more substantial portion of maintenance costs is devoted to accommodating necessary functional changes to the software in order to keep pace with changing user needs. Based on the data we had reviewed, we also noted that systems with well-structured software were more able to accommodate such changes" [Fornell 1992]. Real-time, quick-turnaround maintenance requests rarely include large enhancements done using project management techniques that may take multiple years.

Software-maintenance management should, therefore, do a better job of communicating the many maintenance activities, especially the value-added ones. To do this, it is important that management understand the maintainers' processes and services and their many challenges.

Software maintainers must set up, and better communicate, that there is a fair and efficient queue and priority process in place that manages and monitors the status of each maintenance request/event. This queue management process in software maintenance has many inputs and concurrent interrupting sources, such as: (1) operators who report system failures, (2) users who notice service degradation, (3) development project managers who require current software information and inputs in reengineering studies, and (4) customers who require urgent information. And when there is contention in requests for his or her services, the software maintainer must refer to the SLA and clearly point out the process in place to manage priorities based upon agreed upon service criteria.

## **1.2.2 Maintainers' Perceptions of Software Maintenance Issues**

Maintainers are often confronted with a million lines of somebody else's source code. They must familiarize themselves quickly and process urgent changes without disrupting service. To make maintenance work even more challenging, the new-

ly developed software often has a number of urgent changes pending that the software developers could not include in the initial development, as they had been under pressure to deliver. In a context in which financial resources are limited, the analysis, design, and testing phases of the initial software development project are all put under pressure and are often only partially completed. Such omissions necessarily lead to dire consequences for software maintenance. Maintainers feel that they have little or no control over the quality of legacy code.

It has long been recognized that an immature software development process creates a number of problems in the final product delivered to the customer and to the maintenance organization, whereas a mature process that clearly captures and manages requirements and specifications results a stable design will lower maintenance costs. Use of immature development processes creates an initial maintenance backlog that needs to be addressed, on a priority basis, during the first months/year of operation of the new software. It is observed in industry that processing this backlog creates a large increase in the number of lines of code during the first three years of software maintenance; this is recognized as a major cause of increased maintenance costs and initial dissatisfaction of customers.

Software age also has an impact on maintenance cost when the software was built using older techniques that did not take in account modern architectural concepts, with complex structures, nonstandardized code, and poor documentation, leading to further difficulty in maintenance work. The structure of software undergoing maintenance also becomes increasingly complex as it is modified over time: as the size and complexity of the software increases, an increasing number of resources is required to maintain it.

Software maintenance problems can be grouped into three categories: (1) problems of alignment with the organization's objectives, (2) process problems, and (3) technical problems. Another survey of participants at successive software maintenance conferences presents a list of 19 key maintenance problems, ranked by importance, as perceived by software maintenance engineers (see Figure 1.1). The majority of the problems reported by this survey can be categorized in the maintenance process and management categories (items 3, 7, 8, 9, 10, 13, 14, 15, 16, 18, and 19 of Figure 1.1).

A second group of problems have been identified as originating from the software development process itself (items 4, 6, 11, and 12 of Figure 1.1). When the development quality is poor, the software is still transferred to the maintenance organization with a backlog of changes and problems that should have been addressed by the developers and which are hard to handle with a small number of individuals. Other maintenance problems stemming from the same source are:

- Poor traceability to the processes and products that created the software
- Changes rarely documented
- Difficulty of change management and monitoring
- Ripple effects of software changes

## 6 Chapter 1 Maintenance Issues and Related Management Approaches

Rank	Maintenance problem
1	Managing changing priorities (M)
2	Inadequate testing techniques (T)
3	Difficulty in measuring performance (M)
4	Absent or incomplete software documentation (M)
5	Adapting to rapid changes in user organisations (M)
6	A large backlog of requests for change (M)
7	Difficulty in measuring/demonstrating the maintenance team's contribution (M)
8	Low morale due to lack of recognition and respect for maintenance engineer (M)
9	Not many professionals in the domain, especially experienced ones (M)
10	Little methodology, few standards, procedures and tools specific to maintenance (T)
11	Source code in existing software complex and unstructured (T)
12	Integration, overlap and incompatibility of existing systems (T)
13	Little training available to maintenance engineers (M)
14	No strategic plans for maintenance (M)
15	Difficulty in understanding and meeting user expectations (M)
16	Lack of understanding and support from IS/IT managers (M)
17	Maintenance software runs on obsolete systems and technologies (T)
18	Little will or support for reengineering existing software (M)
19	Loss of expertise when a maintenance engineer leaves the team or company (M)
(M): Management Problem      (T): Technical Problem	

**Figure 1.1** Problems as ranked by software maintainers [Dekleva 1992].

As software development processes progressively improve, there is renewed interest in improving the software predelivery and transition process. This process is requested by maintainers to protect them from taking over unfinished or low-quality software. The predelivery and transition activities address the maintenance-related issues identified during the software development process in order to ensure higher quality which, hopefully, will facilitate the final transition of the software to maintenance. By using such a process, the maintainer ensures that the developer will eliminate the defects before they arise or before time or budget run out to address them. The predelivery and transition process can be considered a quality control process. The lack of such a process can lead to costly maintenance contracts (or, conversely, to very profitable contracts for maintenance contractors).

Many software maintainers also consider that their professional status is perceived as being inferior to that of developers; for instance, they often get into a bind

and have no choice but to accept the newly developed software being forced on them, whatever its quality. The maintainers also report being poorly equipped, supported, and understood by management, while being responsible 24 hours/7 days for the proper functioning and management of software and, more importantly, for the support of all customer-related issues.

Frequent requests for changes by users (items 1 and 5 of Figure 1.1) are also identified as a source of problems; both the business world and the technology evolve rapidly, supported by quick changes in software and also in hardware. Finally, the difficulty in understanding and meeting user expectations is ranked fifteenth.

It is also reported in some organizations that the primary factors contributing to high software maintenance costs are: the number of programmers and their experience, the quality of technical documentation and user documentation, the tools that support maintenance engineers, the software structure and maintainability, and, finally, the contractual commitments that govern a product's maintenance.

Another problem is the impact of the work environment on programmer productivity. It has been reported that the physical environment (noise and interruptions) hinders their performance. Similarly, there is insufficient teaching of software maintenance skills in schools, and what is taught does not always reflect what software maintenance is in reality; this leads to a workforce with a lack of knowledge of maintenance-related processes and techniques.

Is there also a cultural factor that could explain why software maintenance is not sufficiently visible and not promoted? For instance, a study in Japan describes how managers' perceptions, based on the notion that software improvement increases customer satisfaction on a daily basis, seems to be a more important concept in a Japanese organization than it is in a European or American one [Bennett 2000]. Without good software maintenance, these Japanese organizations report that they would rapidly lose market share. When the quality culture is stronger, such as in Japan, maintenance activities benefit from more visibility and they achieve greater recognition: maintenance engineer morale is higher and maintenance management is more visible, both to managers and customers [Azuma 1994].

In summary:

- Nineteen problems have been identified and classified by maintainers, and most of them are managerial.
- Current practices in software-maintenance reporting do not highlight the value-added activities of software maintenance.
- Maintenance specialists' costs are significant and growing.
- There is still a poor understanding of the nature and contribution of software maintenance.
- Software complexity increases and quality decreases with multiple maintenance if no specific action is undertaken to address this issue.
- Low quality development has a direct impact on maintenance costs.

- The local organizational culture and its perception of the contribution of maintenance has an impact on its visibility and on maintenance staff morale.
- There is a lack of automated testing and diagnostic tools for successful change implementation.
- There is a lack of software maintenance training at the undergraduate level.

### 1.3 SOFTWARE MAINTENANCE BODY OF KNOWLEDGE

Every profession has a body of knowledge made up of generally accepted principles. In order to obtain more specific knowledge about a profession, one must either: (a) have completed a recognized education curriculum, or (b) have experience in the domain. For most software maintainers, software maintenance knowledge and expertise is acquired in a hands-on fashion in various organizations. Up to now, very few schools have offered extensive training on software maintenance. Even today, there is still very little in the way of a software maintenance education and training curriculum. Software maintainers and, in particular, software maintenance managers have had access to relatively few books, research papers, or teaching materials, with the exception of the few recent books of Jarzabek [2007]; Khan and Zhang [2005]; Grubb and Takang [2003]; Seacord, Plakosh, and Lewis [2003]; and Polo, Piattini, and Ruiz [2002]. Most other books on software maintenance were published 15 to 25 years ago, whereas generic books on software engineering refer to maintenance only as one life-cycle stage, their main focus being on software development.

The SWEBOK (Software Engineering Body of Knowledge [Abran et al. 2005]) and its Guide constitute the first international consensus developed on the fundamental knowledge required for all software engineers. The 2005 version of the SWEBOK itself can be obtained at <http://www.swebok.org>. In the SWEBOK, there is a knowledge area dedicated to the topic of software maintenance. This confirms that “Maintenance is a specific field in software engineering, and it is therefore necessary to study processes and methodologies which take the specific characteristics of maintenance into consideration” [Basi 1996]. The SWEBOK presents the taxonomy of software maintenance knowledge that is accepted within this specific domain knowledge of software engineering.

The first topic of software maintenance knowledge (see Figure 1.2) is labeled “fundamentals” and introduces the concepts and terminology that form the underlying basis for understanding the role and scope of software maintenance. The subtopics provide definitions and discuss the need for maintenance.

Categories of maintenance work are critical to understanding the underlying meaning of the many types of software maintenance. The second subtopic refers to the “key issues in software maintenance” that must be dealt with to ensure the effective maintenance of software. Software maintenance presents unique technical and management challenges to software engineers (e.g., the challenges inherent in trying to find a fault in a system containing one million lines of code that the maintainer did not develop or does not fully understand). Other examples of software maintenance challenges include the planning of a future release and its develop-

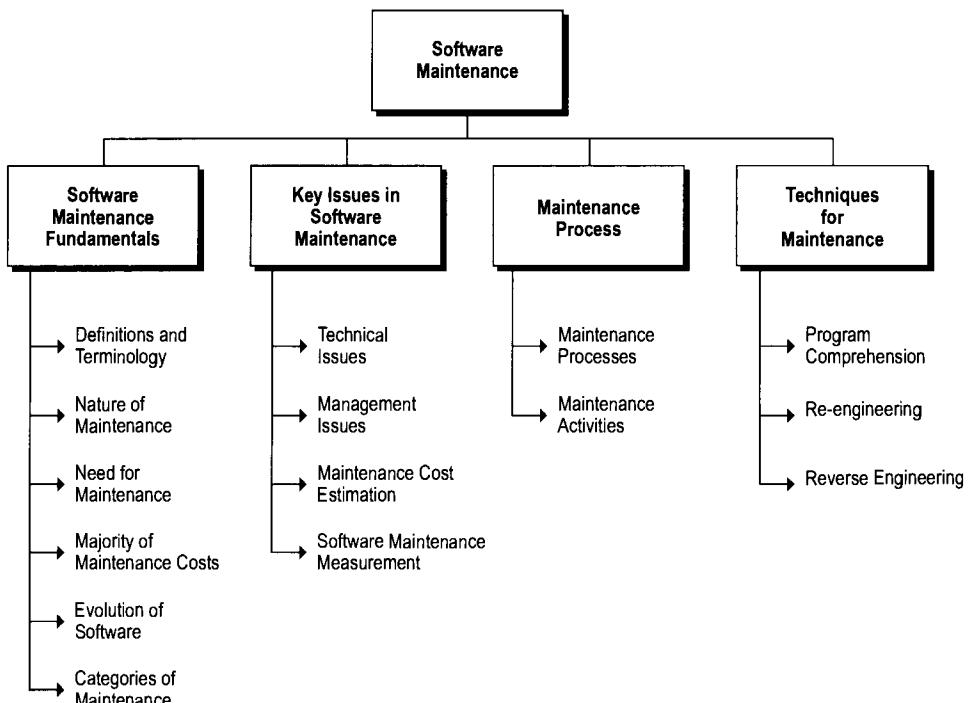
ment in parallel to emergency changes. In the SWEBOK, the maintenance issues are categorized into one of four categories:

1. Technical issues
2. Management issues
3. Maintenance cost and maintenance cost estimation issues
4. Software maintenance measurement issues

The third topic of the SWEBOK refers to the software maintenance process, including maintenance life-cycle models and activities, differentiating maintenance from development, and showing its relationship to other software engineering activities. It is labeled “Maintenance Process” and provides current references and standards used to implement this process.

The fourth topic refers to some of the generally accepted techniques for software maintenance: program comprehension, reengineering, and reverse engineering.

In the following sections, the definition and characteristics of software maintenance are presented.



**Figure 1.2** Software maintenance in the SWEBOK Guide [Abran 2005].

## 1.4 SOFTWARE MAINTENANCE DEFINITION

The software life cycle can be divided into two distinct parts:

1. The initial development of software
2. The maintenance and use of software

Lehman [1980] states that “being unavoidable, changes force software applications to evolve, or else they progressively become less useful and obsolete.” Maintenance is, therefore, considered inevitable for application software that is being used daily by employees everywhere in a changing organization.

Some basic concepts and definitions of software maintenance are now introduced. Software maintenance is often focused on application software in our organizations. For example, application software contains business rules translated into functionality that is used and developed by the business to conduct its daily operations. It is quite different from other types of software that are not under the direct control of the organization or do not contain any business rules (e.g., operating systems and database management systems). We are not saying that these types of software do not require maintenance but it is not the typical business organization that maintains them. Figure 1.3 presents an overview of the often quoted definitions of software maintenance (organized chronologically).

Definition - Interpretation	Author	Year
“Changes that are done to a software after its delivery to the user”	Martin & McLure [Martin 1983]	1983
“The totality of the activities required in order to keep the software in operational state following its delivery”	FIPS [FIPS 1984]	1984
“Maintenance covers the software life-cycle starting from its implementation until its retirement”	von Mayrhofer [von Mayrhofer 1990]	1990
“...modification to code and associated documentation due to a problem or the need for improvement. The objective is to modify the existing software product while preserving its integrity.”	ISO12207 [ISO 1995]	1995
“...the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment.”	IEEE1219 [IEEE 1998a]	1998
“...the totality of activities required to support, at the lowest cost, the software. Some activities start during its initial development but most activities are those following its delivery.”	SWEBOk [Abran 2005]	2005

**Figure 1.3** Generally accepted definitions of software maintenance.

## 1.5 DIFFERENCES BETWEEN OPERATIONS, DEVELOPMENT, AND MAINTENANCE

In an organizational context, the day-to-day maintenance and development of application software developed internally are typically the responsibility of an operational support unit within the software organization. For software acquired externally, the software maintenance is not done by the organization using the software. There is sometimes confusion about who conducts the maintenance and where maintenance activities start and end.

Computer operations activities are considered as distinct from software maintenance activities; for example, the ISO 14764 international software maintenance standard specifies that operational activities (backups, recovery, and systems administration) are handled by the computer operations team, and, therefore, fall outside the standard scope of software maintenance. This scope is similar to what is often mentioned in the literature and, in practice, managers or customers clearly distinguish computer operations from software maintenance activities.

There are still, however, important interfaces between the two, to ensure that the infrastructure supporting the software is functional and efficient (e.g., change management, support calls concerning system failure, recovery of the environment and data in case of disaster, data recovery and work restart, processing time investigations, automated scheduling, and management of disk space and tape libraries) [ITIL 2007b].

Differences between software maintenance and software development activities are sometimes more difficult to distinguish clearly. Software development possesses an interface with software maintenance, which in some organizations is sometimes imprecise and ill-defined. At times, the differences are blurred when the software developer is also involved as well with the maintenance of the software. This blurring is caused by certain maintenance activities that are similar to those found in development (analysis, design, coding, configuration management, testing, reviewing, and technical documentation). Although some activities are similar in theory, they still must be tailored to a maintenance context; for example, small maintenance tasks are being most often performed by one or two programmers with very short-term deadlines and not by a project team with a much longer planning horizon. Significant feature enhancements are rarely done this way as they require a maintenance project approach.

Some authors have studied the differences and similarities between software development and maintenance activities. The main findings are that, in development, the organizational structure mainly takes the form of a project, with a beginning and an end using a team structure to deliver results within an approved budget and time frame. The typical software development project is created for a fixed, temporary period of time, and does not persist past software delivery. The project team develops a plan for resources, costs/benefits, fixed deliverable objectives, and aims for the planned project completion date.

The maintenance organization unit, however, is structured to meet quite different challenges, such as randomly occurring daily events and requests from users, while providing continued service on the software for which it is responsible. Some of the

unique characteristics of software maintenance, as compared to development activities, are [Abran 1993a]:

- Maintenance requests (MRs) come in on an irregular basis, and cannot be accounted for individually in the annual budget planning process.
- MRs are reviewed and prioritized, often at the operational level. Most do not require senior management involvement.
- The maintenance workload is not managed using project management techniques but, rather, queue management techniques.
- The size and complexity of each small maintenance request are such that it can usually be handled by one or two maintenance resources.;
- The maintenance workload is user-services oriented and application-responsibility oriented.
- Priorities can be shifted around at any time, and requests for corrections of application software errors can take priority over other work in progress.

In software maintenance, operational events that cause a software failure are reported in problem reports (PRs). However important the estimated effort may be, PRs are handled by the maintenance team immediately. Whatever the size or effort required to fix a failure identified in a PR, it will be attended to immediately by the maintenance staff. This maintenance service has priority over all other services.

In addition, a user can request a modification to the software by issuing a modification request (MR). The request will go through a number of assessment steps before being accepted. The maintainer will first conduct an assessment to estimate the effort needed to modify the existing software. Dorfman and Thayer's study [1997] indicates that MRs and PRs go through an investigation and impact analysis process that is unique to software maintenance. Impact analysis describes how to conduct a complete analysis of a change in existing software [Koskinen 2005b, Visaggio 2000, Arnold and Bohner 1996]. In the case of the modification, if the estimated effort were too great, the request would be rerouted to a development team to be treated as a small software development project. There is, therefore, in software maintenance, a unique process of acceptance/refusal and classification of work for MRs (see Figure 1.4).

This process takes into consideration as an input the estimated size of the modification [Maya et al. 1996]. April [April 2001] presents the process used in a Cable & Wireless member company, in which the maximum acceptable effort for an adaptive request is five days. This five-day limit is also recognized by the United Kingdom Software Metrics Association (UKSMA):

The distinction between maintenance activity of minor enhancements and development activity of major enhancement is observed in practice to vary between organizations. The authors are aware that in some organizations an activity as large as to require 80 to 150 workdays is regarded as maintenance, while in others the limit is five days. Initially it is proposed that the ISBSG and UKSMA will adopt the convention that work requiring five days or less will be regarded as maintenance activity. [ISBSG 2007]

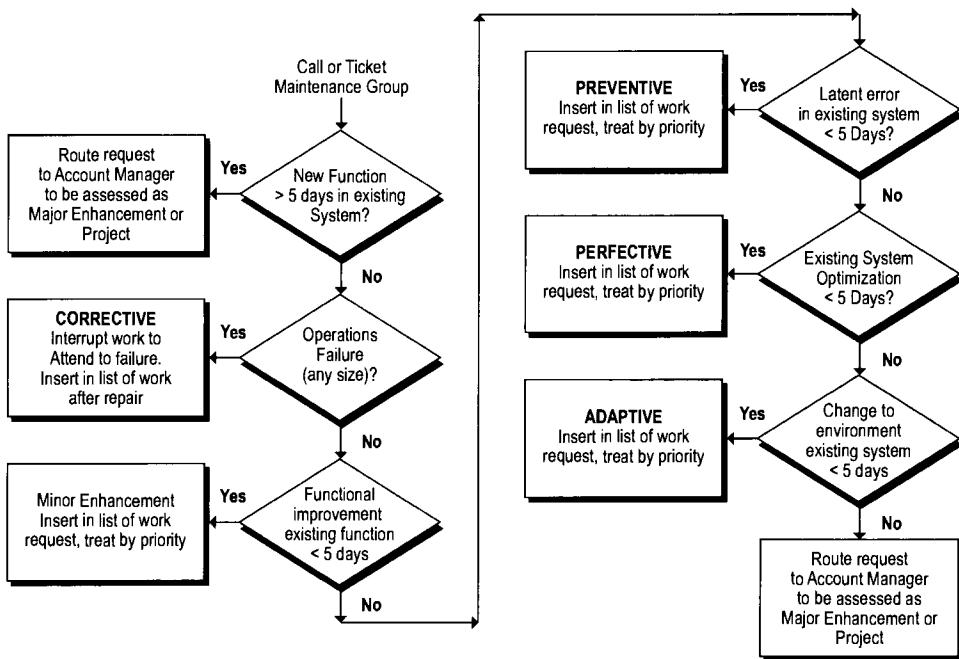


Figure 1.4 Example of the acceptance/refusal process of maintenance work [April 2001].

This very small limit is not likely to exist for maintenance on complex military, nuclear, aerospace, and aircraft software maintenance as it may occur over multiple months or years. The authors acknowledge that they have not taken those industries into consideration when selecting the five-day limit.

Such a threshold is very important in IS/IT organizations, as it dictates when software development starts and when software maintenance ends. For the maturity model proposed later on in this book, no limit in days should be specified. What is essential is that the maintenance work be identified, whatever its estimation effort, and be performed by individuals and not project teams.

Other activities are considered as specific to software maintenance, such as change requests supported by a help desk call center and its support software, activities evaluating the impact of changes, and the specialization in testing and regression checks.

Additional differences in software maintenance have been identified, such as (see Figure 1.5):

- Problem management
- Acceptance of software
- Management of the development's transition to the maintenance team

Some maintenance Activities	Software management (maintenance)	Software development (creation)
Management of problems (problem resolution interfacing with a help desk)	P	A
Acceptance of the software	P	A
Managing transition from development to maintenance	P	A
Establishment of service level agreements (SLAs)	P	A
Planning of maintenance activities (versions, SLA, impact analysis)	P	A
Managing events and service requests	P	A
Supporting daily operations	P	A
Rejuvenating software	P	A

**Figure 1.5** Software maintenance activities (P = present, A = absent).

- Role of the users, the operations team, and the maintenance employees
- Management of maintenance
- Software management (improvements and performance)

Software maintenance can also be described as a service with the following characteristics [Bouman 1999]:

- Emphasis on direct sale to the user
- Frequent and direct contact with the user
- Service supplied immediately, rather than a few months down the road
- Short service time
- The product not always a physical good
- The product not always fit for storage or transport
- Services more specialized and more crafted than physical goods

In summary, software maintenance comprises processes and activities that are not handled by software development groups. Software maintenance also uses processes and activities of software development, especially while implementing a modification to existing application software [ISO 1995, s5.5.3]. It can also be concluded that maintenance is a specific domain of software engineering accepted in the SWEBOK Guide and ISO international standards. It is therefore necessary to examine some of the processes and methodologies that take the specific characteristics of software maintenance into consideration.

## 1.6 WHICH ORGANIZATIONAL UNIT IS RESPONSIBLE FOR SOFTWARE MAINTENANCE?

A software maintainer is defined, in ISO 12207, as an organization that performs maintenance activities. In the industry, software organizations currently favor two organizational structures with regard to the location of the software maintenance function. The first model favors maintenance of the software by its developer. In another model, a software maintenance organization, independent of the developer, takes care of the organization's software maintenance needs. Advantages and disadvantages of each type of maintenance model are described in Abran et al. [2005] and Pigoski [1997].

There are a number of disadvantages to letting the development team maintain the software after it has been put into production:

1. Developers do not like performing maintenance and are more likely to leave for more interesting work.
2. New hires in the development team will be both surprised and dissatisfied to discover that they also need to maintain existing software.
3. Developers are often reassigned to other development projects and prefer this kind of work.
4. When the individuals who developed the software leave, the other employees will probably not be qualified to maintain it.

In management meetings at a Cable & Wireless member company during 2003, it was often mentioned that development organizations that also conduct the maintenance lack transparency and that their products are perceived as being of lower quality, lacking proper documentation, and leaving little flexibility for knowledge transfer between programmers. Pigoski [1994] as well as Martin and McClure [1983] and Swanson and Beath [1989] also concluded that it is in the best interests of large organizations to create an independent software maintenance organization, as they will get better results in terms of quality and independence (e.g., documentation, formalities of transition, specialization of maintenance programmers, management of requests for changes, and overall employee satisfaction). It is not clear, however, at what organization size it becomes preferable to switch to the organizational model in which maintenance is independent of development.

## 1.7 SOFTWARE MAINTENANCE STANDARDS

Software maintenance is considered as one of the five primary processes in the software life cycle processes of the ISO 12207 international standard. In this international standard, the software engineering processes are divided into three main groups: primary, supporting, and organizational. In ISO 12207, the primary processes include acquisition, development, maintenance, and software operation

activities. The support processes include such activities as documentation writing, configuration management, quality assurance, verification and validation, review, audit, and software problem resolution. Finally, the organizational processes are typically offered to the whole organization. These organizational processes include general training, infrastructure, process improvement, and management activities.

In ISO 12207, the software maintenance process includes six major subprocesses (see Box 5.5 in Figure 1.6):

- Process implementation
- Problem and modification analysis
- Modification implementation
- Maintenance review/acceptance
- Migration
- Software retirement

### **5 Primary life-cycle processes**

#### **5.1 Acquisition process**

#### **5.2 Supply process**

#### **5.3 Development process**

#### **5.4 Operation process**

#### **5.5 Maintenance process**

- Process implementation
- Problem and modification analysis
- Modification implementation
- Maintenance review/acceptance
- Migration
- Software Retirement

### **6 Supporting life-cycle processes**

#### **6.1 Documentation**

#### **6.2 Configuration Management**

#### **6.3 Quality Assurance**

#### **6.4 Verification**

#### **6.5 Validation**

#### **6.6 Joint Review**

#### **6.7 Audit**

#### **6.8 Problem Resolution**

### **7 Organizational life-cycle processes**

#### **7.1 Management**

#### **7.2 Infrastructure**

#### **7.3 Improvement**

#### **7.4 Training**

**Figure 1.6** Software maintenance as a primary process of ISO/IEC 12207 [ISO 1995].

The maintenance process may also call on other primary, supporting, or organizational processes when needed. This standard also clarifies which of the activities, also shared by developers, should be adapted when used by maintainers (i.e., documentation, configuration management, quality assurance, verification, validation, reviews, audits, problem resolution, process improvement, infrastructure management, and training).

The ISO 12207 standard also suggests that the maintainer can adapt other software engineering subprocesses that come from:

- A development process
- A management process
- An infrastructure process
- An improvement process
- A training process
- A supply process

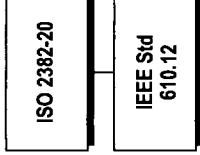
ISO 12207 does not, however, provide guidance on how to do this; more specific guidance to software maintenance engineers is provided in the ISO 14764 international standard.

What clarifications are provided by ISO 14764 in regard to the relationship between maintenance and development (ISO 12207's *primary process*)? The international software maintenance standard ISO 14764 specifies that when development standards are being used by maintainers they must be adapted to meet the specific requirements of maintenance [ISO 2006, s8.3]. The more specific references in ISO 14764 and ISO 12207 are as follows: in ISO 12207, it is notably specified that the analysis used to determine which part of the software and documentation will be modified (s5.5.3.1) must use development processes, and section 5.3 describes the development process associated with the documentation of the test, test results, and test review activities that must also be called upon (s5.5.3.2).

The ISO software engineering standards (i.e., ISO 14764) are oriented toward “classic” activities and do not address certain aspects of the processes that support software maintenance as experienced in industry. Additionally, a significant number of standards are used by software maintainers. The most relevant IEEE and ISO standards, as well as their interdependencies, are presented in Figure 1.7. James Moore explains in his book, *The Road Map to Software Engineering: A Standards-Based Guide* [Moore 2005], the layered diagram he developed to show the relationships between different standards “from the general to the more specific.” We have used the same technique here to come up with a similar representation from the software maintainers’ perspective. James has kindly agreed to review it for us.

In this representation, we see that maintenance could utilize other software engineering processes quoted in at least 40 other standards to conduct its daily activities. So, how can we explain that the majority of maintainers do not currently use software maintenance element standards and other referenced standard processes? One tentative explanation could be that it is partly because the maintainers “inherit” not

## Terminology



18

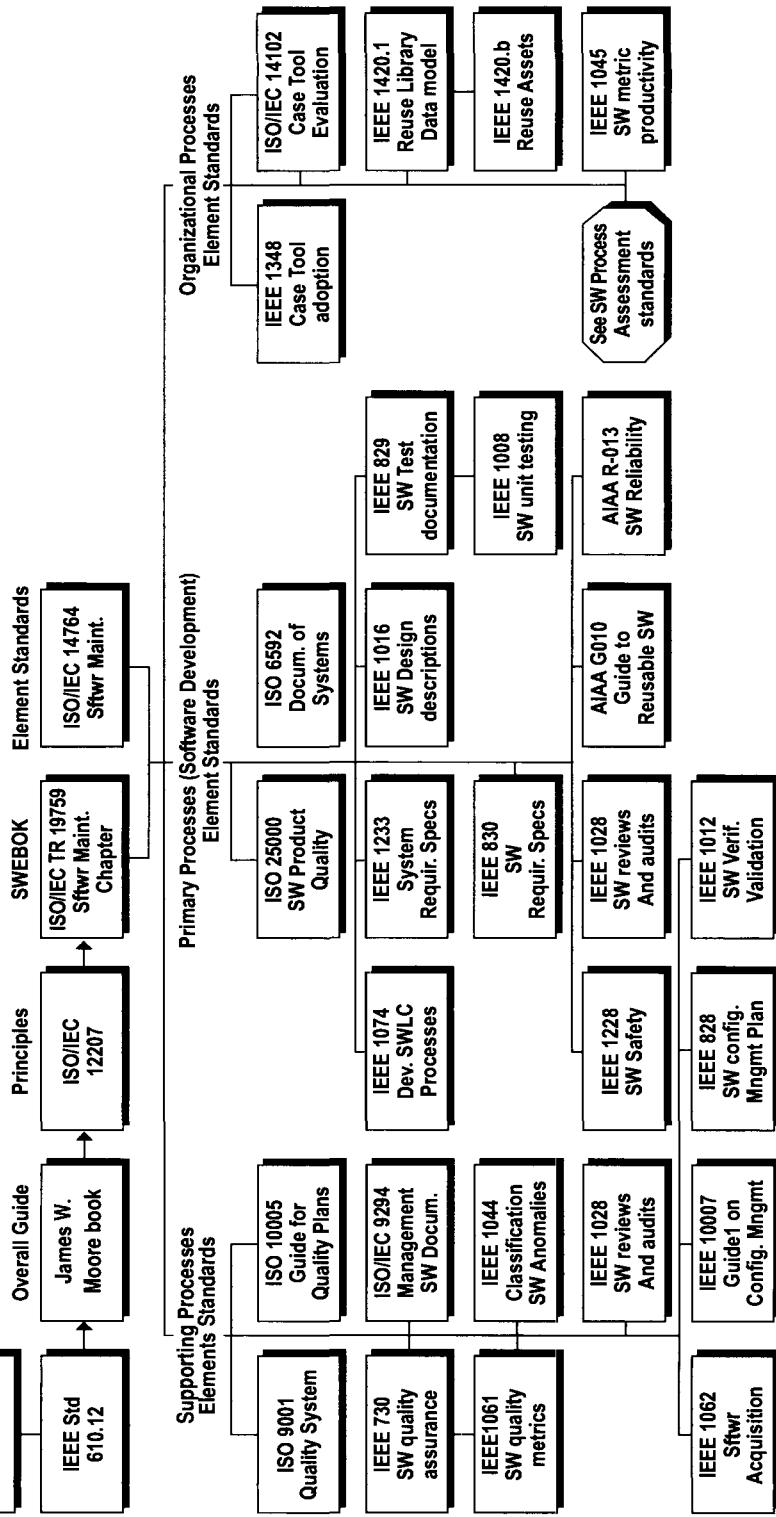


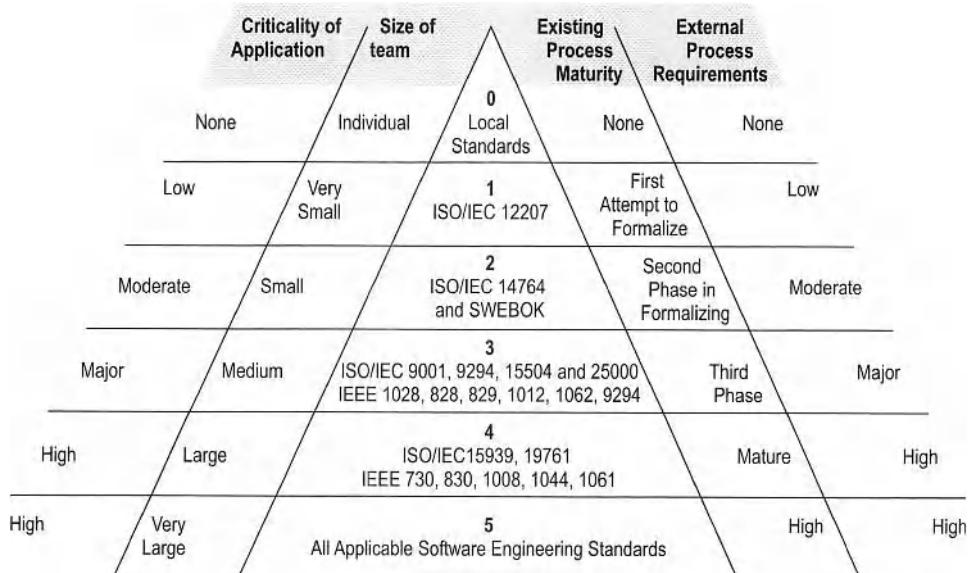
Figure 1.7 Software standards that could be of use to software maintainers.

only the legacy code but the legacy processes to go with it. Hence, the maintainers tend to follow the “standards” set forth in the legacy code. Another tentative explanation is that most maintenance organizations demonstrate all the characteristics of a low-maturity-level organization and are mostly reactive rather than proactive in the management of software maintenance.

On the other hand, a few organizations at the highest CMMi maturity level would use standards extensively, such as in the space shuttle organization, in which the team has been together for years, and technology does not change much.

With such a plethora of candidate standards, Schmidt, in his book, *Implementing the IEEE Software Engineering Standards* [Schmidt 2000], argues that it is not realistic to consider using all available standards right away when implementing formal processes in a software organization. Schmidt argues that organizations will adopt a progressive use of software engineering standards, suggesting that there may be a direct relationship between an organization’s maturity and its use of standards. He, therefore, proposes a pyramid representation that describes the progressive use of standards based on four factors: (1) the application’s criticality level, (2) team size, (3) the maturity of existing processes, and, finally, (4) external process requirements (see Figure 1.8). We have used this representation to show what type of pyramid we may get from a software maintainer’s point of view.

In this model, the processes become more and more rigorous as the criticality level of the software increases. Similarly, as the size of the teams increases, more and more standards will be needed in order to establish proper support for increasingly important processes and internal communications. An organization’s maturity



**Figure 1.8** Applicable standards pyramid—a simple maturity model.

will also have an impact on the number of standards that can realistically be integrated into existing employee operational processes without being rejected/resisted. Finally, external requirements can enforce the use of certain standards when the products do not conform to certain industry-specific regulations (e.g., nuclear, medical, aeronautical, or defense).

Level 0 describes the adoption of very high-level processes and is characterized by the absence of formal standards. It is only possible to be on this level if there are no external requirements and the software has low criticality. At this level, it is possible to describe the maintenance work using mostly the ISO 12207 standard, which allows for the presentation of a general model of software activities. This ISO 12207 standard does not delve into the details of how activities are accomplished in a specific organization. The other five levels of the standards pyramid are:

Level 1 standards are used for the study of requirements and the documentation of tests.

Level 2 progressively adds formality by introducing work plans, documentation of concepts, and the use of a systematic approach for reviews.

Level 3 goes even further by proposing the use of standards for quality assurance, verification and validation, configuration management, and security issues.

Level 4 proposes additional standards for the life cycle, configuration management, unit testing, and two important standards for measuring software quality.

Level 5 proposes the use of all currently published IEEE standards.

Schmidt's model could apply to software maintenance (because software maintenance directly refers to activities for the development process in ISO 12207. His model is a theoretical proposition, in the sense that it is supported by neither experiments nor implementations; it still provides an expert opinion in support of maturity model concepts being potentially supported by the many software engineering standards.

Some have argued that that maintenance processes reflected in current international standards approximately match level 2 practices of the SEI model. It is also suggested that a direct relationship exists between the maturity of an organization and its progressive use of standards concepts.

In summary, a large number of software standards can be referenced by maintainers. One international standard is central to software maintenance (ISO 14764). The software maintenance domain refers to software development standards when needed. Maintainers must, therefore, use software development standards while making sure to adapt them to the specific maintenance context [ISO 2006 s8.3.2.1 and s8.3.2.2]. Incidentally, by nature, standards do not include topics on emerging technologies or those that are not perceived as generally accepted. The list of unique software maintenance activities are candidates for inclusion in a future version of the ISO 14764 standard as the members of the ISO/JTC1/SC7 committee accept them as generally accepted.

## 1.8 SOFTWARE MAINTENANCE PROCESS AND ACTIVITIES

Although software maintenance international standards have been available for more than 20 years, many organizations still do not have a well-defined process for software maintenance activities. Van Bon [2000] has also reported that there is little in the way of software maintenance process management and that the domain continues to be neglected.

In the early days of the software industry, there was no difference between software development and software maintenance. In the beginning, legacy code was so small you could rewrite it every time there was a change needed! And programmers did it for fun. Only in the 1970s did life-cycle models specific to the software maintenance process start to appear. In general, these first models were broken down into three steps: (1) understanding, (2) modifying, and (3) validating changes to software. Although many development methodologies still do not represent the software maintenance step, others have included software maintenance as their last step. From the software-life-cycle process point of view, the IEEE 1074 standard considers maintenance to constitute the seventh of eight phases of the development project cycle, at the end of the implementation phase of a new piece of software. However, some have suggested that “the traditional view of the software development cycle has done a great disservice to the maintenance domain by representing it as one step at the end of the cycle” [Schneidewind 1987].

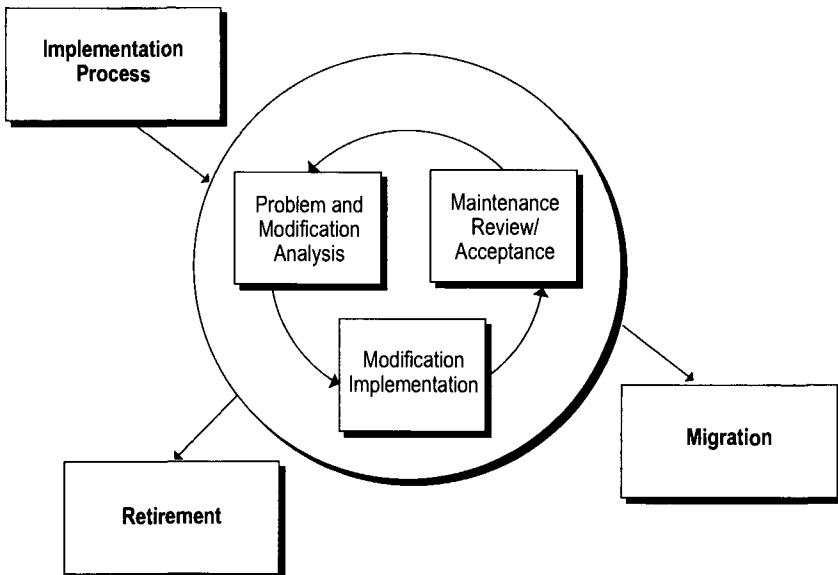
The 1980s brought models oriented toward the maintenance process [Bennett 2000, Fugetta 1996]. Maintenance was no longer seen as the last step in development. These models present maintenance-specific activities and introduce precedents for each of them. Some consulting organizations have also defined their own maintenance standards, which comprises variants of maintenance-specific activities.

The 1990s saw the emergence of the international standards (ISO 12207 and ISO 14764]) that are being used today. In its current version, the ISO 12207 standard represents maintenance processes that overlap many other phases of the software-life-cycle processes, in the sense that its realization may require going through, partially or totally, most of the software-life-cycle processes. To better situate and present the boundary of the software maintenance processes in ISO 14764, a general diagram is presented in this standard (Figure 1.9).

The ISO 14764 process model includes the six key processes of software maintenance. Individual organizations and suppliers can, of course, adapt this process model, which must generally meet the international standard.

Let us look at each process in more detail. The implementation process is external to the more operational processes (shown inside the circle) of software maintenance. It contains software preparation and transition activities, such as the design and documentation of the maintenance plan, the preparation for handling problems identified during development, and the follow-up on product configuration management.

There are three operational processes within this process model, as shown inside the circle in Figure 1.9:



**Figure 1.9** Software maintenance key processes [ISO 2006].

1. The problem and modification analysis process, which is executed once the application software has become the responsibility of the maintenance group. The maintenance programmer must analyze each request, confirm it (by reproducing the situation), check its validity, investigate it, propose a solution, document the request and the solution proposal, and, finally, obtain all the required authorizations to apply the modifications.
2. The modification implementation process considers the implementation of the modification itself. This process is more of a development-specific process (with a direct reference to the development process activities in ISO 12207, section 5.3).
3. The maintenance review/acceptance process for the acceptance of the modification. This consists of a verification and approval with the individual who submitted the initial request, and ensures that the solution provided meets the initial requirements.

Migration processes (platform migration, for example) are less frequently used, and are not part of the daily maintenance tasks. If the software must be ported to another platform without any change in functionality, these processes will be used and a maintenance project team is likely to be assigned to the specific project task.

Finally, the last maintenance process, an event which does not occur on a daily basis, is the retirement of a piece of software. Both the migration and retirement processes are typically handled as projects, not as small maintenance work, since the effort usually requires that thresholds be set for work classified as small mainte-

nance requests. From this perspective, some activities of software maintenance are quite different from daily software support.

As discussed elsewhere in this chapter, a considerable number of the maintenance activities mentioned in the literature have not yet been standardized and, therefore, are not represented in the process model in Figure 1.9 but are observed in industry.

## 1.9 SOFTWARE MAINTENANCE CATEGORIES

In 1976, E. B. Swanson [Swanson 1976] was one of the first to come up with a three-category classification of maintenance activities, which he labeled corrective, adaptive, and perfective. This was followed by a survey that classified maintenance work into four categories: corrective, adaptive, perfective and preventive. Today, ISO 14764 still uses these maintenance work categories (see Figure 1.10).

More detailed categories of maintenance work have since been proposed recently [Chapin 2001]. Other approaches, like ontology, have also emerged to describe further the software maintenance processes and activities. An ontology (see Figure 1.11) represents a view of a domain in which the concepts are defined in a precise and sometimes formal way. A number of publications present concepts of ontology applied to software maintenance [Kitchenham 1999, Derider 2002, Vizcaino 2003, Dias et al. 2003, and Ruiz 2004].

## 1.10 MAINTENANCE MEASUREMENT

Measurement is fundamental to Deming's view of process control: "A process is stable if its future performance is predictable within statistically established limits" [Deming, 1986]. By measuring, we can learn more about the behavior of resources, the process, and the software product, and their relationships of cause and effect (see Figure 1.12).

### 1.10.1 Maintenance Process Measurement

A process is a sequence of steps performed for a given purpose. It is generally accepted that the quality of software is largely determined by the quality of the development process used to design it. The maintenance manager's objective, then, is to

	Correction	Enhancement
Proactive	Preventive	Perfective
Reactive	Corrective	Adaptive

Figure 1.10 ISO 14764 software maintenance categories.

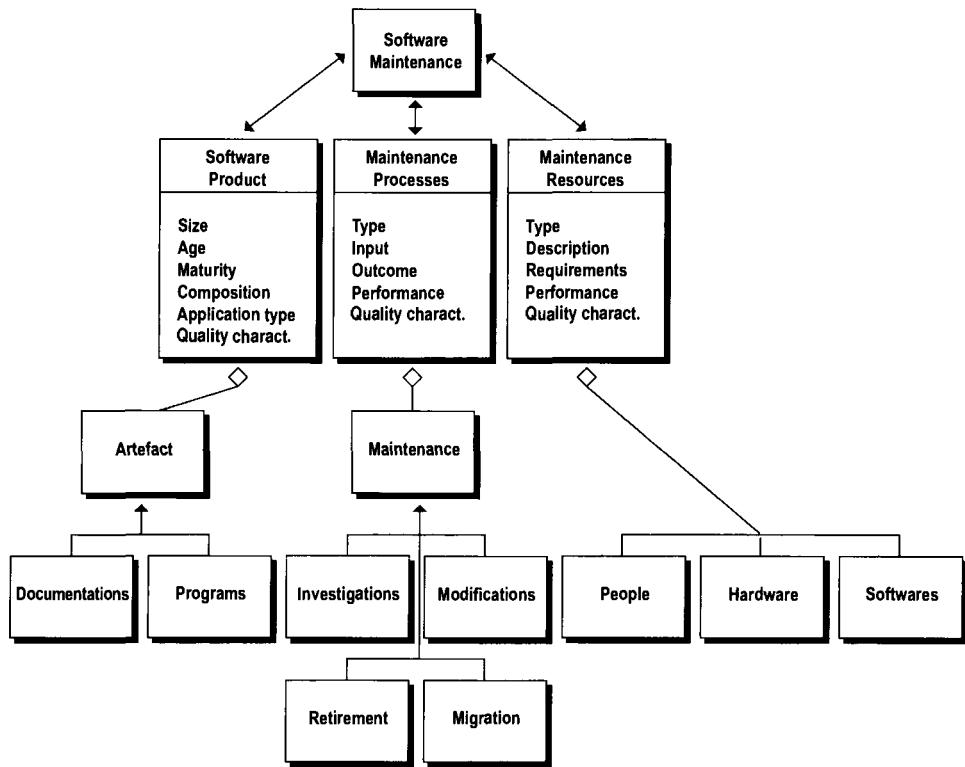


Figure 1.11 Ontology of software maintenance [Kitchenham 1999].

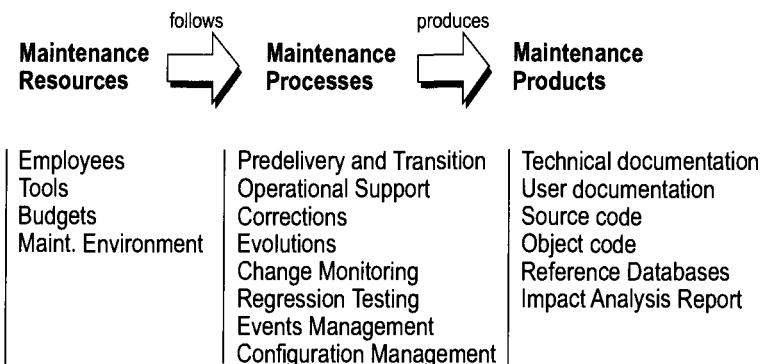
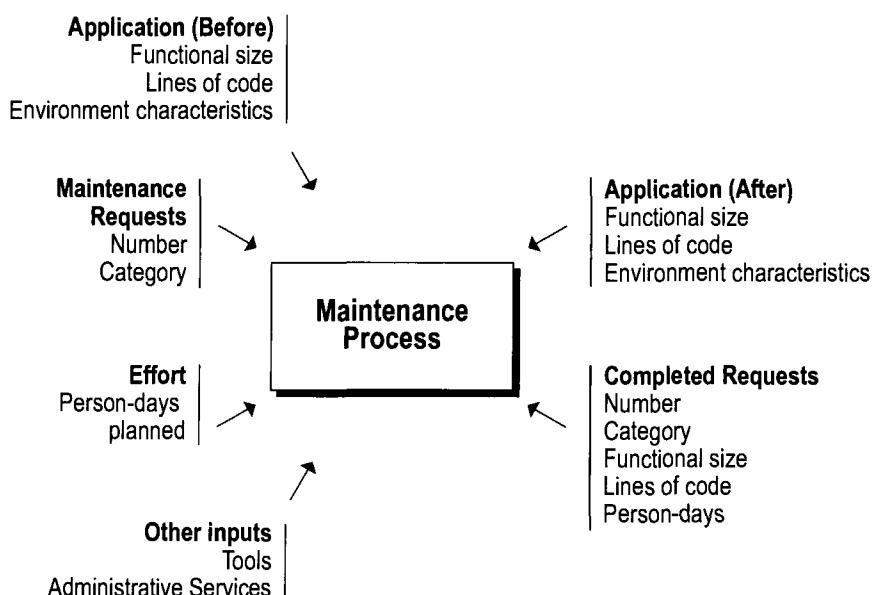


Figure 1.12 Software measurement perspectives.

help bring such a process under control, and measurement has an important role to play in helping to meet this objective. Because he or she has little control over the development of the software, the maintainer must identify the earliest point at which it is possible to influence the maintainability characteristics of the new software under construction. Initiating measurement during predelivery and transition could be the strategy to use to assess the quality of the product and its readiness to be accepted in maintenance. To achieve this goal, a decision can be made to implement a software maintenance measurement program and link it to the software development measurements. For example, if the maintainer can set some maintainability objectives early on in the development of new software, its quality could be measured both during and after development. For all of us, this would be an ideal situation. At the highest maturity levels, a maintainer would be in that situation. We can only hope that the rising maintenance costs will motivate organizations to improve considerably their maintenance processes.

Many factors must be taken into account before measuring software maintenance processes. One strategy is to identify the key activities of a given process and their improvement goals. These key activities, in turn, have many characteristics that can be measured. Desharnais and coworkers [Desharnais et al. 1997] show an example of the software maintenance characteristics they chose for a pension plan agency in Canada (see Figure 1.13). They recommended that measures be well de-



**Figure 1.13** Example of selected characteristics of a maintenance process [Desharnais et al. 1997].

fined and verified to ensure that they properly characterize the product, the services, and the maintenance processes. But before measures are well defined, it is essential that processes and products be defined first.

Maintenance measurement prerequisites have been presented in April and Al-Shrougi [2000], together with requirements in terms of (1) quality goals, (2) definitions of work categories, (3) implementation of request management process/software, (4) classification of maintenance effort in an activity account data chart (billable/unbillable), (5) implementation of activity management (time sheet) software and data verification, and (6) the measurement of the size of requests for change.

Many organizations often rely too much on qualitative information from internal reviews to evaluate software quality when they should be using quantitative methods instead. “Low maturity organizations often collect measures just to show data. It is rarely used in practice” [Ebert 2004 p. 174]. The SEI CCMi describes process measurement activities as appearing at level 2 maturity and refers to the necessity of a basis of maintenance process before measurement can be initiated.

The measures proposed by the SEI were suggested from a development perspective and do not capture the peculiarities of software maintenance. Other authors [McGarry 1995, Desharnais et al. 1997] also confirm this view and specify that a software maintenance measurement program must be planned separately from that of the developers. The measurement requirements being different, software maintenance measures are more focused on problem resolution and on the management of change requests.

High maturity organizations established maintenance measurement programs as early as 1987. Such a measurement program at Hewlett Packard is documented in Grady and Caswell [1987, p. 247], illustrating that measurements must be based on well-documented software processes. Grady and Caswell highlight that a process that is not documented, is difficult if not impossible to improve using measures. Hewlett Packard implemented a company-wide measurement program, including a description of the collection of data specific to software maintenance. Grady and Caswell identify the following concerns that are specific to a software maintenance measurement program:

- How should we plan for staffing maintenance of a software product family?
- When is a product stable?
- Is mean time between failures (MTBF) for a software product really a meaningful measure of software quality?
- In which development phase should tool and training investments be made?
- How does documentation improve supportability?
- How many defects can be expected in a project of a given size?
- What relationships exist between defects found prior to release and those found after release?
- What, if any, is the relationship between the size of a product and the average time to fix a defect?

- What is the average time to fix a defect?
- How much testing is necessary to ensure that a fix is correct?
- What percentage of defects is introduced during maintenance? During enhancements?

“Estimation is still one area in software measurement where many struggle mightily—right from the start” [Ebert 2004]. Software maintainers are no different in this respect. The estimation of software maintenance resources (effort, number of employees, and budget) is typically conducted by software maintenance engineers using one of two approaches [Abran and Maya 1995, Sellami 2001]. The most common estimation approach in software maintenance is based on experience, that is, individual recollection as a basis to determine maintenance requirements, extrapolate their impacts, and estimate expected costs. Another approach consists of using maintenance-specific parametric models (such as the COCOMO maintenance model [Boehm 1981] or similar proposals [Hayes et al. 2004, Chan 1996]). However, there are few maintenance organizations that report using these tools in their daily work.

Experience is often used as a basis for estimation in software engineering, as there is a lack of quality historical data. The following paragraphs illustrate some types of measurements that are needed in software maintenance and the need to compile historical data that could be used to build software maintenance estimation models.

A number of authors present specific measures that are useful to maintainers. In a survey on the measurement of maintenance, Dutta and coworkers [Dutta et al. 1998] report that 75% of individuals surveyed indicated that they documented the number of failures, analyzed their cause, and identified the origin of failures in production software. Another ratio of interest is the *average change time*, which is the average time it takes to implement a change in application software after its initial development; application software with a shorter average change time is considered to have better maintainability.

Hitachi reports using a maintainability cost measure (e.g., cost of changing software after its development) by using the ratio of the number of failures to the cost of the initial development project. By measuring multiple maintenance projects from the same organization, Tajima [1981] could then document measures to identify which application software has better maintainability than others.

Authors [Abran 1991, Abran 1993a, St-Pierre 1993, Desharnais et al. 1997] also report individual measures and software maintenance portfolio measurement grouped by software maintenance category. They identify trends within each of these categories and used these trends to explain the maturity profile of software.

Maintainers performing maintenance of the Space Shuttle software have lots of experience on the software they maintain, have worked with the same team for years, and have not upgraded the technology of their maintenance process for a long while. This is the kind of environment that operates at a high level of process maturity. NASA proposed 12 measures for corrective and adaptive software maintenance [Stark et al. 1994]: “application size, number of employees, number of re-

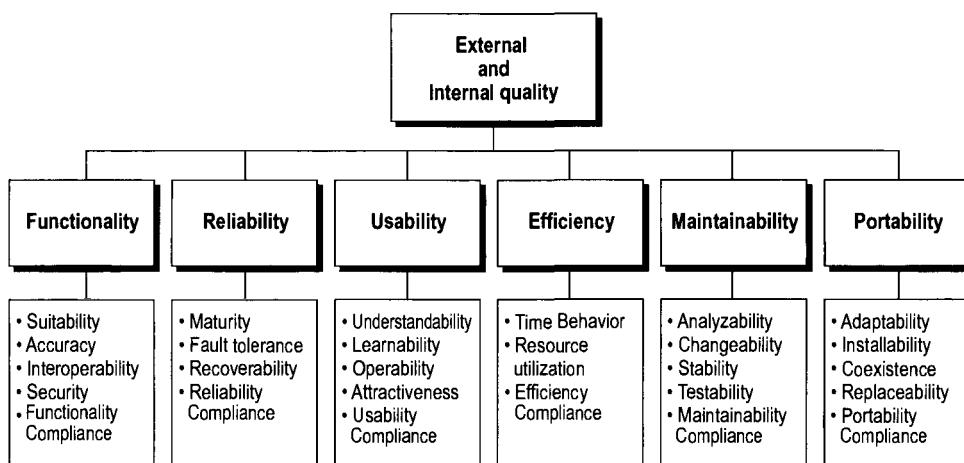
quests/status, number of perfective requests/status, use of computer resources, bug density, volatility, issue report open time, failure/repair ratio, reliability, design complexity, bug type distribution.”

### 1.10.2 Software Product Measurement

The quality of a software product is of particular importance to maintenance teams. The ISO 9126 [ISO 2001] standard identifies six characteristics of a software product’s quality, including maintainability (Figure 1.14). This standard is currently under review by Work Group Six of ISO/JTC1/SC7. The maintainability characteristic is further described by four subcharacteristics: analyzability, changeability, stability, and testability. “Maintainability is not restricted to code; it describes many software products, including specifications, design, and test plan documents. Thus we need maintainability measures for all of the products we hope to maintain” [Pfleeger 2001].

Two different perspectives of maintainability are often presented in the software engineering literature. From an external point of view, maintainability attempts to measure the effort required to diagnose, analyze, and apply a change to specific application software. From an internal product point of view, maintainability is related to the attributes of application software that influence the effort required to modify it. The internal measure of maintainability is not direct, meaning that there is no single measure for the application’s software maintainability and that it is necessary to measure many subcharacteristics in order to draw conclusions about it.

The IEEE 1061 [IEEE 1998b] standard also provides examples of measures without prescribing any of them in particular. By adopting the point of view that reliability is an important characteristic of software quality, the IEEE 982.2 [IEEE



**Figure 1.14** ISO 9126—Software product quality model [ISO 2001].

1988] guide proposes a dictionary of measures. This standard identifies, among other things, six distinct measures of source code complexity.

Internal measures often describe the structural complexity of software. Structural complexity measures are generally extracted from the source code using observations on the program/module/functions graph of software source code. Studying graphs allows for collecting information on the complexity of the application software. There is a large body of literature on the static evaluation of source code. These studies are based on work carried out in the 1970s by McCabe, Halstead, and Curtis [McCabe 1976, Halstead 1978, Curtis 1979]. Since then, a number of researchers have proposed a large number of measures more adapted to an object-oriented environment.

Maintainers, managers, and especially users are looking for a single-number approach that indicates the relative maintainability of a given software; they basically wish to use this number to help in the decision to accept/reject the transition of a new software product from development to maintenance. For instance, some have proposed a maintainability index, which is a composite measure incorporating a number of traditional source-code measures into a single number [Welker 2001]. In practice, however, the single-number approach is quite elusive and has been found to be of use at best as a very rough indicator. Much further research is still required in this area.

There are a number of software tools available in the industry to measure source code complexity. These include ready-to-use measures and also tools that allow the user to create new ones for specific requirements. However, interpreting these measures is very difficult since they are very specific and there are few mechanisms to summarize the data for decision making. Managers and customers often end up with technical measures that do not add much value to be communicated to the software user.

In summary:

- The measurement of software products is a specialized domain; the ISO 9126 measures were published in the early 2000s and maintenance organizations still have to put them into practice.
- Tools to measure quality of source code are available commercially.
- Interpreting source code measures is still difficult, since they are very specific and few mechanisms exist to summarize the data for decision-making purposes.
- Subjective measures still play an important role in assessing software maintainability.

## 1.11 SERVICE MEASUREMENT

Some authors claim that “software maintenance has more service-like aspects than software development” [Niessink 2005]. This seems to be the case for other information technology services as well, like computer operations. Service quality mea-

sures for software maintenance services have been proposed in the literature and divided into three categories:

1. Internal service-level agreement
2. Maintenance service contract
3. Outsourcing contract

### 1.11.1 Internal Service-Level Agreement

To reach agreement on service levels, a consensus must be developed between the customers and the maintenance organization about the related concepts, terms, and measures to be used. Service-level agreements (SLAs) are said to be internal when they are exercised entirely within a single organization. This type of agreement documents consensus about activities/results and targets of the many maintenance services. SLAs originally appeared in large computer-operations centers during the 1950s. They progressively extended their reach to all IS/IT service-oriented activities during the 1970s [ITIL 2007b]. However, many IS/IT organizations still do not have SLAs in place today [Karten 2007].

Some attempts to identify exemplary practices and to propose SLA maturity models have appeared recently. COBIT® [IT Governance Institute 2007] is a set of exemplary practices used by IS/IT auditors. COBIT® identifies and describes practices that must be implemented in order to meet the IS/IT auditor requirements for SLAs. It identifies the *definition and management of the service level* as an essential practice, with the measurement of quality of service as its main objective. COBIT describes five maturity levels for the agreement (nonexistent, ad hoc, repeatable, defined, managed/measured, and optimized). COBIT® also describes other software engineering processes that are directly related to software maintenance. Other proposals of SLA maturity models that have recently been put forward can be found in Niessink et al. [2005] and Kajko-Mattsson et al. [2004].

SLAs become an important element of customer satisfaction in a competitive environment. A few publications have directly addressed SLAs in a software maintenance context [Bouman 1999, COBIT 2007, Niessink 2000, April 2001, Niessink et al. 2005, Kajko-Mattsson et al. 2004].

Even though many publications have presented detailed elements of SLA, there is still little known about the key underlying principles at work in these types of agreements. Bouman [Bouman 1999] presents various concepts that have been proposed to organize the knowledge about the structure and use of an SLA. One such concept is that an SLA should be based on results rather than on effort. For example, in many instances, attempts are made to describe the results of an IS/IT service in terms of quality levels to be achieved for the IS/IT service, and with the expectation that this is to become the basis for assessing whether or not the terms of the SLA have been met. This agreement is called a results-based SLA, and is an approach which requires that a consensus be reached on the quality levels and that some quantitative measure be assigned to every service and product. IS/IT organi-

zations, including some outsourcers, are known to avoid this path and claim all sorts of technical hurdles to the implementation of this type of agreement. It has been observed in practice that where there is no consensus on a proper description of results; the fall-back position is to describe only the effort and costs to be spent on maintaining the software. These are called effort-based SLAs. However, effort-based SLAs cannot be easily controlled.

Another concept proposed by Bouman is an inventory of guideline templates to be used for the specification of an SLA document between two parties. One should not underestimate the difficulty of establishing service measurement. Additionally, the SLA should clarify the expectations/requirements of each service. In the context of an internal agreement on software maintenance services between the maintenance organization and its users/customers, two attitudes have been reported [April 2001]:

1. On the one hand, the customer wants to concentrate on his business and expects a homogeneous service from his IS/IT organization. The result of this homogeneous service for the customer is the ability to work with a set of information systems without any disturbance whatever from the source of the failure. The way in which this service, these systems, or their infrastructure are constituted is of less interest to the customer, who would like this vision to be reflected in his SLA. This means that a result-based SLA on the total service (including help desk and computer operations) should be described, and not only on the individual/partial IS/IT components (i.e., a server, a network, or software).
2. On the other hand, software maintainers only own a portion of the service as perceived by the customer. They are often quite ready to provide a result-based SLA for their services. But the products are operating on infrastructures that are not under their responsibility (desktop, networks, and platforms), as they only interface with the help desk and computer operations. In [April 2001], the internal SLA of a Cable & Wireless member company is described in detail.

To achieve integrated measurement of all the components of the software requires the involvement of all groups concerned, and requires that someone in IS/IT must own an overall SLA in which all the IS/IT services are described. All the IS/IT service organizations that support the customer must be documented in a unified SLA to satisfy the customer. A unified SLA is one that includes the service levels of all the IS/IT organizations that are involved in supporting the end users. Included are the topics that are more maintenance-specific in a unified SLA:

- Responsibilities of the maintenance customer
- Responsibilities of the maintenance organization
- Description of maintenance services:
  - Maintenance program management
  - Management of requests, priorities, and the request management software

Corrective, preventive, adaptive, and perfective maintenance  
Planning and management of software versions (releases)  
Configuration management  
License management, escrow delivery, and contracts with third parties  
Recovery after disasters  
Customer support  
Exclusions  
Detailed list of supported software, by priority  
Service fees  
Service hours  
Escalation procedures in case of problems  
Measures of performance  
Reviews and mechanisms for solving disagreements and conflicts

A software maintenance SLA requires a clear definition of service, measurements, and objectives, and needs a documented and detailed inventory of the software that is supported. For each software product, the following topics are to be documented: its customers, specific responsibilities, platforms, support hours, historical work volume, equipment rental and rental customers, availability objectives, parameters of recovery in case of disaster, third-party contracts, and the list of maintainers assigned.

The current approach to SLAs still suffers from a number of problems [Bouman 1999]:

- The customer does not want to be involved and expects homogeneous service from his IS/IT organization. Technical details about services and software products are not very important to him as he is more focused on functionality of the software.
- The description of the expected maintenance results are not written in a way that is readily understood by the customer, but is rather complex and often technical.
- There is confusion in the transfer of responsibilities between the development and maintenance organizations. The concept of the original software warranty is difficult to establish in this context.
- Service rewards and penalties are generally not part of the agreement.

### **1.11.2 Maintenance Service Contracts—External Service Agreement**

Maintainers also require a second type of contract knowledge for contracts involving third parties, who are often contracted to conduct maintenance on software on an annual basis. These contracts are called maintenance service contracts. The soft-

ware industry makes a distinction between a license and a maintenance service contract. In the industry, a maintenance contract will typically include software updates and support activities for use should a problem arise.

A good way to negotiate the maintenance service contract and license price is to act early during the initial purchase of a software. It is often too late if the acquisition or development team has already finalized the initial purchase/project agreement and contracts. The organization that acquires the software is then locked in and there is no supplier/developer incentive to give reductions or additional services for maintenance services. An important rule to remember is that maintainers must be included in the predelivery negotiations in order to use the initial contract/project discussions to ensure that the maintenance-related issues are included clearly and completely, and that maintenance services will provide value for money.

Observing maintenance services contract negotiations between buyers, suppliers, and maintenance managers in the industry makes it possible to better understand the typical maintenance agreement content. Below is an outline for such an agreement:

- Definitions
- Supplier obligations
- Maintenance services and optional services
- Scope of maintenance
- General terms of the agreement
- Customer and customer personnel obligations
- Confidentiality
- Reproduction of documentation and source code
- Limits of responsibility, acts of God, order payments, survival, laws
- Support procedures

The consensus between the two parties regarding the classification of errors and the procedures that they will have to use to report a problem are typically documented in the contract appendices. Maintenance categories, requests for changes, problem severity, response time, and escalation are also typically defined there.

There are still some specific risks inherent to maintenance service contracts:

- The supplier proposes temporary solutions to problems instead of permanent ones.
- The supplier will attempt to negotiate renewals of the license and the maintenance separately in order to make more money on both.
- Additional charges are not clearly defined.
- Escrow services are rarely discussed or included in the costs.

Mastering the maintenance service contract domain requires that a number of maintenance engineers be trained and assigned to software supplier negotiations

(during predelivery activities). Some websites discuss and present software maintenance contract clauses and provide hints that could help businesses keep up with suppliers' business strategies in this area [Business Link 2007].

In summary, the maintenance service contract domain is a specialized one, and it has been the focus of very few publications up to now. The manager and the maintenance engineer are typically at the mercy of the suppliers if they do not acquire some specialized knowledge on how these contracts are structured and negotiated.

### 1.11.3 Outsourcing Agreements

A third type of service agreement is the software maintenance outsourcing contact. This contract moves the software maintenance to a third party for a period varying from 5 to 10 years. An outsourcing contract is often a global agreement with an IS/IT supplier or an industry leader with an important foothold in the IS/IT sector. It is typically long-term, and once it is in place the organization will sever ties with both its former personnel and the application software itself. In this type of agreement, the outsourcer will take charge of the personnel and will propose service agreement clauses.

The main justifications for outsourcing software maintenance are [Carey 1994]:

- Promises of decreasing costs
- Access to the expertise of the outsourcer's personnel
- Move from a fixed-cost structure to a variable-cost structure
- Collect revenue from the sale of an asset
- IS/IT not being one of the company's strategic activities
- Transfer of technical details and problems to the outsourcer

A typical outsourcing contract is quite different from a service contract. For instance, the outsourcer takes over the application software, as well as the source codes, and treats software packages separately (SAP/R3, Oracle HR, etc.) from the general agreement:

- The outsourcer offers typically a 90-day warranty following the correction of a specific problem. If the problem has not been fixed to the client's satisfaction, the outsourcer provides additional maintenance services free of charge.
- The outsourcer asks the client to determine the priority of maintenance work items.
- The outsourcer keeps a list and a history of problems submitted using help desk software.
- While the ISO 14764 software maintenance work categories are not often used, the following work categories have been often observed: corrections, preventive, regulations, release control, and ad hoc reports.

- A reference to an internal service agreement and measurement is made, but no report, measures, or review process is formally proposed.
- The duration of the outsourcing agreement is between 5 and 10 years.

McCracken [2002] reports that 50% of companies become involved in outsourcing without having a clear service agreement, and he also refers to a report from the Gartner Group that places this figure at 85%.

In summary, outsourcing contracts documents contain financial objectives, but do not yet completely define outsourced services (according to international standards) and targeted service levels (with reports and precise measures).

## 1.12 SOFTWARE MAINTENANCE BENCHMARKING

A popular definition of benchmarking was originally reported by David T. Kearns, CEO of Xerox: "Benchmarking is the continuous process of measuring products, services and practices against the toughest competitors or those companies recognized as industry leaders."

Three types of benchmarking were proposed by Xerox. The first is internal benchmarking, which aims to compare different sectors in the organization. For software maintenance, an internal comparison allows one to draw conclusions for improvement. The second type, competitive benchmarking, requires data from direct competitors. This approach is made difficult by problems with accessing information from other organizations and, thus, understanding how these competitors perform in their chosen functional area. The last proposed approach is functional benchmarking, which consists in finding organizations that perform well in the same functional area, but in other industries. For general maintenance of hardware equipment, the methods used by NASA, nuclear power plants, and the aerospace industry can be examined to find useful information.

What about the functional benchmarking of software? A prerequisite for carrying out a benchmarking exercise is a clear understanding of internal processes. Although it is not necessary to have a very elaborate measurement program in place, there must be data on hand. Some argue that benchmarking could be a waste of time if the organization does not possess that understanding and a certain number of internal process measurements.

Two approaches are more common in the software industry. The approach most often observed is competitive benchmarking with the participation of an outside benchmarking service provider specialized in computer services benchmarking. For example, some benchmarking organizations use their own commercial approach to collect and analyze data, based on their own corporate questionnaire for data collection. A few weaknesses were noted in this approach by the managers we interviewed on the topic:

- Short time period (2 months) and difficulty of collecting validated data (often the requested data was not available and had to be made up on the fly)

- Comparative data is hidden (it is unknown against whom the comparison is being made)
- Problems with the quality of data (e.g., quality of the count of the lines of code)
- Credibility of the lines of code to function points conversion ratios
- Number of resulting diagrams/graphs (40 comparative viewpoints)
- The interpretation of results does not take into account specific facts but is rather made with generalizations that are hard to contradict or support
- Difficult to get detailed information about the best performers and what they do to get this kind of performance

In summary, a good number of measurements are presented to the organization, but it is difficult to know whether or not they are valid, or to find out the practices of those companies that claim to perform better.

A number of risks of the same type have been reported. Firms that use this approach specifically for maintenance present, at the end of the exercise, variants of the following graphs/measures:

- Function points supported per person (in-house development)
- Function points supported per person (in-house development + software packages)
- Cost by supported function points
- Average age (in years) of application software (currently in production)
- Age groups (% by functionality) of application software (currently in production)
- Number of supported programming languages
- Supported data structures (sequential, indexed, relational, others)
- % of programming types (maintenance, improvements, new applications)
- Support—environment complexity index (based on the number of users, data size, platform size, and power)
- Support—technical diversity (number of applications, programming languages, data archiving technologies, tools, and operating systems)
- Support—use of CASE tools
- % of personnel stability (turnover rate)
- Duration of employment (years)
- Human resources level (% of total company employees)
- Salaries
- Training effort (number of training days per person per year)
- Ratio of faults in production (by criticality: critical, major, minor, or cosmetic) per 1000 supported FPs
- Ratio of faults in production (by cause category: design, programming, environment, or other) per 1000 supported FPs

A second approach is benchmarking within a user group. The best-known ones are the IT Benchmarking User Group and the International Software Benchmarking Standards Group [ISBSG 2007]. The advantage of these organizations is that they, under some conditions, allow their members access to their databases and thus enable the sharing of experiences. Participation in this type of activity is specialized and does not include the purchase of a “ready-to-use” service. Therefore, only a few organizations will participate in these user groups because they require a local champion dedicated to this task for a number of years. It has been, however, observed that up until now such organizations had little data in software maintenance but rather focused on software development projects.

Capers Jones mentions a last kind of benchmarking which he calls “Assessment Benchmarking,” which he compares to a medical evaluation of the IT organization in order to establish a diagnostic on what works and what does not using an industry guide of best practices [Jones 1994]. The SEI’s model is the most widely known and used for this type of benchmarking. In the software industry, this exercise is considered as a process improvement exercise rather than as a benchmarking exercise.

In summary, software benchmarking does not easily allow scientific comparison because the data is neither controlled nor validated. However, benchmarking is sometimes done too quickly and the conclusions do not allow for the clear identification of lessons for improvement from benchmarked organizations. Benchmarking activities with open databases are now feasible but require sustained help from a knowledgeable champion. Organizations in other, often more mature, industries publish insightful results from the use of this benchmarking practice.

## 1.13 SUMMARY

In this chapter, many maintenance issues have been presented from two perspectives (internal and external). Maintainers are aware that customers perceive these issues through quality of service and costs, both of which are key drivers of customer satisfaction.

Also presented was the perception of software maintenance engineers and managers, documenting the reasons why they claim it is so difficult to maintain software. A large number of these problems are management related and occur in a constantly changing environment. Moreover, many of the difficulties encountered in maintenance have their origin in the initial development of the software. This highlights the importance of involvement in predelivery and transition activities; with other partners involved, maintainers must be able to actively participate in early project and contract negotiations to ensure that their voice is heard.

Software maintenance is a specific domain of software engineering and is now included in the SWEBOK as a main knowledge area of software engineering.

Two standards were central to software maintenance for many years (ISO 14764 and IEEE 1219). A project within ISO JTC1/SC7 was done in 2005–2006 to harmo-

nize the two standards. In 2006, a new version of ISO 14764 was published and IEEE1219 was withdrawn. However, it is important to note that international standards do not include all the maintenance activities observed in the industry today and inventoried in many recent publications.

Software maintenance and software development have indeed some activities in common. Maintenance refers to specific software development activities, listed clearly in international standards. The standards also indicate that maintainers must make sure to adapt them to the specific maintenance context and not just use them as the developers intended.

This chapter has also highlighted that measurement plays an important role in assessing maintenance process and product quality. Maintainers have specific processes, as well as specific measurement programs and approaches. Products can also be assessed for their quality by specifying maintainability measures (external and internal maintainability).

It has further been demonstrated that maintainers need contract knowledge in order to ensure that they can influence software quality, build internal agreements with their customers, and manage actively their suppliers.

To conclude, software maintenance is a unique domain and a maturity model that includes the specific characteristics of maintenance is worth designing.

## 1.14 EXERCISES

1. You have been promoted to the position of software maintenance manager. Meanwhile, there is a heated discussion about developer/maintainer interface problems. The development manager is lobbying to bring this issue under his control and, in this way, put an end to these problems which impact clients without any obvious benefits. You must prepare to discuss this situation at a senior management meeting that will be attended by your colleagues in software development. Draw up your own proposal and present the pros and cons of your colleagues' position.
2. Describe the unique characteristics of software maintenance.
3. It has been pointed out to you that, over time, the development teams have established fine-tuned development processes, and that it would be economical and useful to reuse these processes for maintenance purposes. In which standards can you find pointers about development processes that can be of use in software maintenance? Describe your proposed strategy for applying these standards to maintenance, and use them to support your case.
4. Your organization's position is that maintenance activities belong at the end of the software development life cycle. However, this contention contributes to marginalizing your work, and does not fully represent all the maintenance you perform. What changes can you propose to the software life-cycle documentation to provide a more comprehensive view of soft-

ware maintenance activities? What standards could be of use to you for this purpose?

5. There are two of you working in software maintenance. Your boss requires that you follow ISO standards. How can you achieve this without spending all your weekends at work?
6. The standards do not cover all the maintenance activities performed in your organization. Describe five activities that do not appear in ISO 14764. Document the source of your information and explain why, in your opinion, these activities are not included in this international standard.
7. Operations services phoned you this morning and asked you to carry out checks to ensure that the software will function correctly on the new network. Explain (1) in which maintenance category you classify this work, (2) what kinds of tests are necessary, and (3) do you have to request authorization from your customers before starting work?
8. This week, your customer sends you a third request that entails a significant modification. He is new to his job. Explain to him, in simple and clear terms, the differences between software maintenance and software development.
9. Your boss tells you that, during the last negotiation meeting with the customers, they did not accept the proposed increases in service charges. Help your boss by proposing an alternative solution based on the various maintenance categories.
10. Describe some types of preventive requests in software maintenance.

# Chapter 2

---

## Maturity Models in Software Engineering

### This chapter covers:

- Basic concepts and terminology in software process maturity models
- Differences between a maturity model, a quality standard and an evaluation method
- How to design a maturity model
- The validation of a maturity model
- An inventory of maturity models

### 2.1 INTRODUCTION

To implement an industrial-strength production process, the development of a controlled and repeatable process is necessary, and procedures are of paramount importance for successfully implementing such a process. Nelson and coworkers use the term “routine” to refer to “predictable behavior models which are observable within organizations, from well defined technical procedures to more general policies and strategies” [Nelson et al. 1982]. Without such routines, it is not possible to produce goods on a large scale using only materials and human resources, and any attempt to do so would result in what is basically a craft process.

On the one hand, an organization is said to be immature if its routines are not predefined, but rather redefined on the spot by its staff and management. Immature organizations are reactive and manage events on a case-by-case basis. This leads to continual surprises that threaten production times and product quality. The performance of immature organizations then relies strictly on individual expertise that concentrates, and limits, the knowledge required to perform and to manage the workload: “When production relies on the availability of specific people, the organization does not establish any fundamentals for the general long term improvement of productivity and quality” [SEI 1993b].

On the other hand, an organization is said to be mature when its routines are well defined and communicated to its employees, such implemented routines being then recognized as operational [Humphrey 1991]. It should be noted that individual expertise generally takes a long time to acquire, and is often derived from many years of trial and error, a fairly costly way of finding an individual practice that yields the expected benefits. Once these routines are known and documented, it becomes possible to both teach them quickly and objectively analyze them in order to improve them.

To leverage this knowledge about the different levels of maturity, it is important to understand how to move up across maturity levels. For the past 10 years in the software domain, there has been a growing interest in process evolution, and the pursuit of solutions to software-related problems has led to the development of reference models for the evolution of software processes to higher maturity levels. The pioneers were from the general domain of quality (including Deming, Juran, and Crosby), who postulated that the quality of a product is directly related to the quality of the process designed to develop it and to provide postdelivery services. A mature process does not necessarily mean a quality process. Maturity simply provides a mechanism to measure how well the process works. Within this paradigm, it is considered that the best way to improve software quality is to improve its development and maintenance processes. In the maintenance context discussed here, the term “process” refers to all activities, methods, techniques and tools used in maintaining software.

To move progressively from an individual chaotic process to a more mature one that is well structured and deployed throughout the organization, an evolution path is most useful; such an evolution path is usually described by a maturity model. If a maturity model does not provide information on the steps required to move from one level to the next, the risk of failure in improvement programs is considerable, the management team not having the roadmaps, concepts, or knowledge of the fundamentals needed to understand how and where to direct the improvement efforts.

## 2.2 OVERVIEW OF BASIC CONCEPTS (PROCESS AND MATURITY)

The concepts (and terminology) used in developing maturity models are introduced in this section. The Software Engineering Institute’s report by Ibrahim [Ibrahim 1995] presents an overview of the basic concepts of process maturity. The key terminology used in the domain of improvement using maturity models is as follows:

*Process.* “A process can be defined as a set of activities that transform inputs into outputs and thereby fulfill a purpose. Typically the outputs from one process become the input(s) to one or more further processes. The activities operating within a process should be coherent and complete” [TICKIT Project Office 2001].

*Software process.* “A set of activities, methods, key practices, and transformations that people use to develop and maintain software and its associated products” [SEI 1993b].

*Key practices.* “The activities which contribute the most in a process” [SEI 1993b].

Processes typically work at three levels within an organization:

1. At the organizational level, they set general and corporate objectives.
2. At the organization’s tactical and operational levels, they set project and functional goals (for a specific organization).
3. At the personal level, they influence specific task activities.

Processes can be characterised in terms of their capability, performance, and maturity:

*Process capability.* “The range of expected results which can be obtained using a specific process” [SEI 1993b].

*Process performance.* “The observable results of the use of a specific process” [SEI 1993b].

*Process maturity.* “The scope and granularity with which a specific process is explicitly defined, managed, controlled, measured and efficient” [SEI 1993b].

The terms “capability” and “maturity” are often used interchangeably in the literature. The correct use of these two terms is the following: the maturity of a process is evaluated in order to establish its capability; the capability of a software process has the following characteristics [Grant 1997]:

- It varies from one organization to another.
- It cannot be bought or instantly acquired.
- It must be developed incrementally within the organization.
- The accumulation of wealth (assets) is not enough to improve it.
- It cannot be outsourced.
- Its evolution is continuous and dynamic.

The SEI proposes a *model for the maturity* of software processes. In this context, the maturity model is defined by the SEI as a representation of key attributes of selected organizational activities that have an effect on its progression toward the full realization of its potential [Garcia 1993].

The concepts of software process evolution and of related key attributes are embedded within this definition. This leads to the introduction of the concept of the staged representation, in steps or levels, for these key attributes, to ensure that im-

provements achieved at a given step can be leveraged for further improvements for the next step. Crosby's quality management evolution grid is the source of the SEI's design of maturity steps or levels. Crosby identified five consecutive steps in quality management: (1) uncertainty, (2) awakening, (3) enlightenment, (4) wisdom, and (5) certainty [Crosby 1979].

In addition, this continuous improvement approach relies on a planned number of small improvements over time rather than on unpredictable breakthroughs.

To develop such a model, classification and abstraction work on knowledge about software processes has been carried out at the SEI. Figure 2.1 describes the initial concepts in the model developed by Feiler [Feiler 1992]. This facilitated the development of consensus at the SEI on definitions and models of the fundamental knowledge expected of software organizations to develop application software.

Feiler explored first the identification of scopes and key practices for each concept category in Figure 2.1. It is at this point that an architecture for *process areas* was introduced into process models: (1) "process scope," (2) "process roadmap," and (3) "key sectors."

A *key process area* is a set of key practices pertaining to the same general domain of activities. Maturity models then sort key practices into logical groups. It can be observed that there are

- Five in the ISO 15504 part 5 assessment model: customer–supplier, engineering, project, support, and organization
- Four in CMMi: process management, project management, engineering, and support
- Nine in the Camélia model [Camélia 1994]: management of quality, resource improvement, processes, management, quality system, development practices, customer support, and administrative data processing

These groupings then lead to the concept of process domains, including key process areas and roadmaps for improvement. A process domain is a term used for the ma-

Process Definition Structure	Process Engineering	Process Implementation	Process Properties
Architecture	Development	Execute	Static
Design	Adaptation	Control	Dynamic
Definition	Planning (deployment)	Define the roles	
Planning (resources)	Instantiation	Conformity	
	Evolution		

Figure 2.1 Process concepts [Feiler 1992].

ajor grouping of processes in a maturity model. Process domains are divided into areas called key process areas. Some maturity models will subdivide key process areas into road maps. A *road map* is defined as a set of linked practices that can often cover many levels of maturity. In a given roadmap, the order of key practices is determined by a consensus of experts who agree on their sequencing and degrees of priority in the mastery of a specific process.

It is also observed that the more fundamental key practices have been positioned at the lowest maturity level, whereas the more difficult key practices are positioned at the highest maturity level.

From these findings, a *process model* can be derived, and can be described as “a documented description of the practices that are considered fundamental to good management and engineering of a business activity [ESI 1998a].” It defines how these practices are combined into processes that achieve a particular purpose.

The process maturity evolution approach stems from work on total quality, which proposes a voluntary and continuous improvement, and not necessarily the specific prescription of a standard. By contrast, standards prescribe a specific sought-after process state.

## **2.3 DOES CMMI COVER SOFTWARE MAINTENANCE ADEQUATELY?**

The literature review on software maintenance has confirmed that some maintenance processes are unique to maintainers and not part of the software development function. This situation motivates why software maintenance process models like the S3<sup>m®</sup> are needed. Specifically, maintenance processes tend to be subjected to inappropriate maturity evaluations in our organizations today. With its primary focus on project management, CMMi does not explicitly address the issues specific to the software maintenance function. For example, in CMMi

- The concept of maintenance maturity is not recognized or addressed.
- There is insufficient inclusion of maintenance-specific practices as process improvement mechanisms.
- Maintenance-specific issues are not adequately addressed.
- Rejuvenation-related plans such as the need for redocumentation, reengineering, reverse engineering, software migration, and retirement are not satisfactorily addressed.

This was also observed in the previous version of the model, the CMM, in 1995 by Zitouni [Zitouni 1995]. The above items are still absent from the new CMMi version, since it maintains a developer’s view of the software production process. This means that, whereas assessments and improvements of large maintenance activities requiring a project management structure could use CMMi, evaluations and improvements to small maintenance activities that do not use project management techniques would benefit from using maturity models better aligned to their specif-

ic characteristics. When these unique maintenance processes are compared to the CMMi model content, it can be observed that CMMi does not explicitly address these topics. This will be further developed in Section 3.10.2.

More recently, Craig Hollenbach has been assigned as project manager to design the CMMi for services. The CMMi for services will be based on ITIL and will address IT services, for example, desktop management, network management, and operations and maintenance (O&M—operating and maintaining a delivered system and its operational environment). The current CMMi model is appropriate to use for the development of typical systems, and software-engineered products as well as for service systems (e.g., network engineering, help desk systems, manual or automated service aids). Yet, the current CMMi model does not cover the provisioning or delivery of services, including the management of performance-based services, the availability of service components, the dynamic handling of service requests, among other things.

## **2.4 DIFFERENCES BETWEEN MATURITY MODEL, QUALITY STANDARDS, AND EVALUATION METHOD**

It is often challenging to grasp the differences between a quality standard, a maturity model, and their evaluation methods. ISO 9001:2000 is an international de jure standard to establish a quality management system, whereas the CMMi can be called a de facto standard aimed at software development improvement. In the software engineering community, both the CMMi and ISO 9001 are well known.

### **2.4.1 Choosing Between ISO 9001 and CMMi**

It must also be noted that, in general, ISO 9001 is much better known than CMMi. Never before has an international standard benefited from such broad global acceptance. The ISO 9001 standard is relevant to all industries, whereas CMMi is relevant only to the software industry. At first glance, CMMi and ISO 9001 appear to be very different. First, these models use completely different terminology. Second, the ISO 9001 standard is described at a fairly high level (i.e., it has only 23 pages compared to the 643 pages of CMMi).

Another significant difference between the two approaches to quality must be noted: the patent exclusivity rights and commercial openness are not the same. For example, the ISO 9001 model is universal, does not include copyright fees, and each country can set up its own certification mechanisms. By contrast, the SEI's model is controlled by a single organization that controls the related training and certification processes. To offset this exclusivity problem, a number of countries have worked, through ISO committees, to develop the ISO 15504 international standard. Within this international consensus, to which the SEI conforms, it is possible to develop other maturity models. Ideally, we really need fewer maturity models and more documentation on the “hows” of adapting existing software engineering standards to specific situations.

A striking similarity between the two documents is their shared interest in process quality and improvement, although the standard does not specify how to execute the improvement. Models such as CMMi, however, are quite explicit on this subject.

The expected benefits of ISO 9001 certification have been extensively documented:

- Improvement of internal documentation
- A better quality product
- Competitive advantage
- Fewer customer audits

Manufacturing companies have been well informed of the usefulness of quality standards. They learned that, in a competitive market, independent proof of conformity could give them a competitive edge in acquiring new contracts and clients. In this context, a number of buyers explicitly require the supplier organization to have its ISO 9001 certification before giving out a contract.

In parallel, the U.S. Department of Defense requires proof of conformity to CMMi level 3 from all its subcontractors [SEI 1993a, p. A-1]. Gradually, these two conformity requirements (ISO 9001 and CMMi) have become a requirement in winning business in the software industry.

What has become important over the decades for manufacturing companies to achieve has not yet necessarily become so for service industries. Software maintenance, being much more like a service industry, has more difficulty interpreting and implementing these new standards and models. Service organizations experience greater difficulty dissociating the final product from the processes used in its production.

Compliance with the key concepts of such standards requires that they be adequately interpreted for a specific industry. Additional guidance has, therefore, been developed for the implementation of ISO 9001 in service industries, including the software industry. There are now two recognized sources for interpreting the ISO 9001:2000 standard according to the specific characteristics of the software industry:

1. ISO 90003:2004—a guide to interpreting the ISO 9001 standard for software
2. Interpretation material (customer guide) developed by [TICKIT 2001] for ISO 9001 auditors who need to tackle the software domain

These guides clarify how each ISO 9001 clause has to be applied in an audit of an organization that designs, maintains, and operates software. It is interesting to note that both the CMM and CMMi also have their own rules of interpretation.

In any context in which a process model is applied, a reasonable interpretation of the practices should be used.

There has been a convergence of the CMMi and ISO 9001 models with the recent release of the ISO 9001:2000. The changes that bring the two together are the following:

- The use of a process approach is now required in both.
- The continuous improvement approach for quality system improvement is now common to both.
- There is more attention to the role of management in both, including a commitment to develop and improve processes.
- Both contain specific references to fundamental principles in quality management.

The new version of ISO 90003 refers to the standard describing the software life cycle processes (ISO 12207) and, therefore, draws a direct parallel between software life development activities and each ISO 9001:2000 clause.

Organizations that use their ISO 9001 certification assets as a stepping stone toward meeting CMMi requirements benefit from an important advantage. Dache [2001] states that level 3 of the CMM is reachable for an organization that already has ISO 9001 certification. There is also the emergence of a trend toward the use of the two models concurrently to meet common objectives of CMMi for software and ISO 9001.

In summary, the software industry now recognizes the importance of the ISO 9001 standard and of maturity models such as the CMMi for improving software processes. These types of complementary models are used to fulfill the commercial need of an organization to differentiate itself from its competition and the need to improve software processes. Professional and business judgements are required for a wise use of these quality models in audit and evaluation.

### 2.4.2 The Evaluation Method

This section presents the differences between the evaluation methods used with ISO 9001 and the ones used with maturity models.

The quality of the software processes of an organization can be assessed using standards or using maturity models. Jones [Jones 1994] compares a maturity evaluation to a medical exam of the IS/IT group to find out what is working well and what is not. In the software industry, this exercise is also referred to as either benchmarking or as a diagnostic step before a process improvement activity is initiated.

To obtain an ISO 9001 certification, an organization must submit itself to a quality audit. An independent registrar, who can vouch for the organization's quality system, performs the audit. Figure 2.2 presents a subset of recommended evaluation methods used with ISO 9001 and with maturity models (used for evaluation and audit activities).

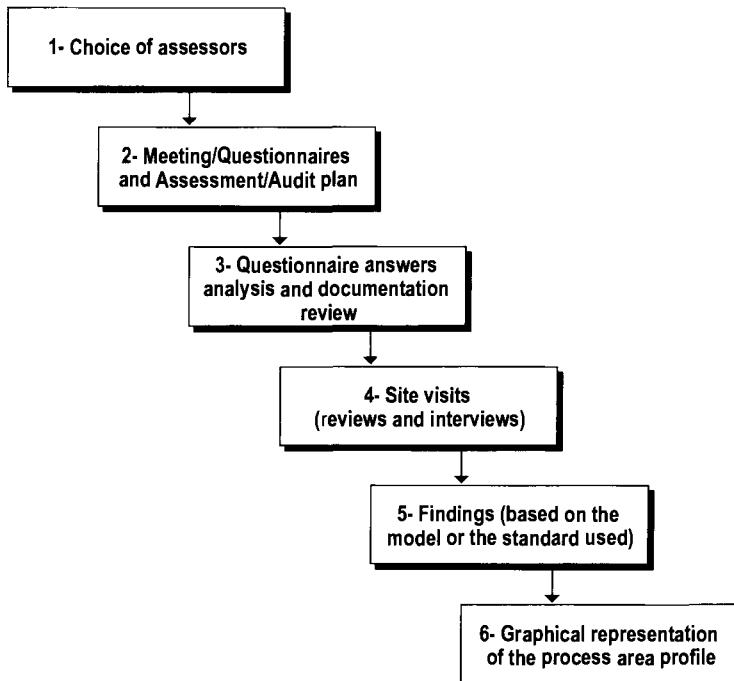
The maturity model evaluation methods and ISO 9001 audits have similar foundations, and they, therefore, share many steps in common. Figure 2.3 presents a model of the major steps required to deliver an unbiased status report on an organization's processes at a specific time, using a maturity model.

The first step is the identification and selection of the person or people who will be part of the evaluation team. These people must have (or obtain) the necessary

Quality standard	Maturity models	Evaluation methods
ISO9001:2000 standard [ISO 2004a]		ISO 19011:2002 [ISO 2002a]
	ISO15504-2 standard [ISO 2003]	ISO15504-3 [ISO 2004c]
TickIT ISO9001:2000 interpretation guide [TickIT 2001]		ISO 19011:2002 [ISO 2002a] ISO 17011 [ISO 2004e]
	CMM model for software [SEI 1993b]	CBA-IPI [SEI 2001b] SCE [SEI 1996b]
	CMMi model for software [SEI 2002]	SCAMPI [SEI 2000]
	FAA-icmm model [FAA 2001]	FAM [FAM 2006]
	CMM for systems engineering [SEI 1995]	SAM [SEI 1996a]

[1] Currently in version 2.0 (2001)

**Figure 2.2** Examples of evaluation and audit methods.



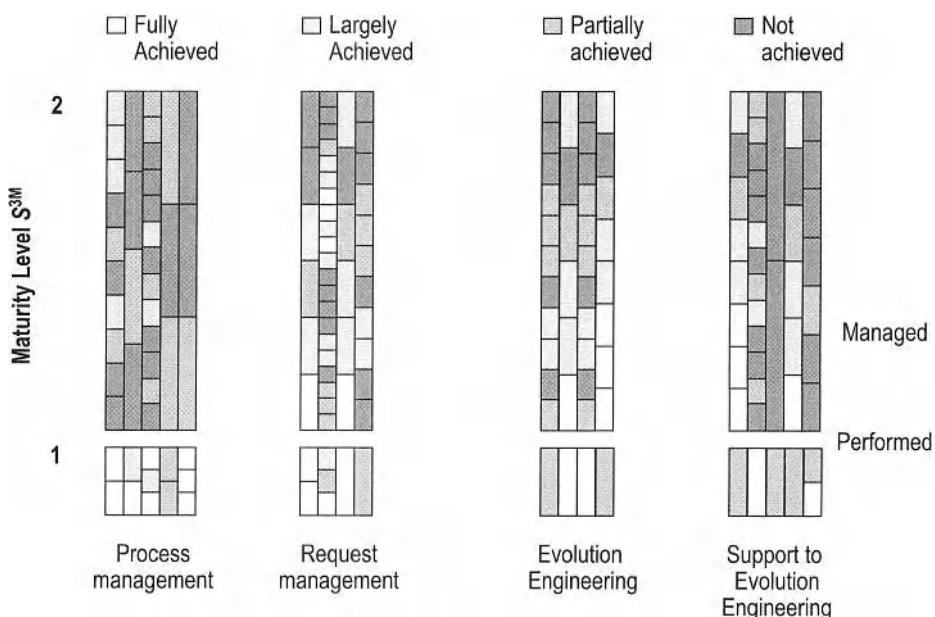
**Figure 2.3** Major steps in evaluation methods.

training and, in some cases, the required certifications to perform the evaluation. The second step is the initial contact with the organization to be evaluated, using either a survey instrument or a planning meeting to explain how things will be done, to validate the scope of processes, determine the number of people to be involved, and to draw up a general agreement on the evaluation plan. The third step is a summary of planning meetings results and, sometimes, of the survey, to figure out where to start and to identify specific avenues for questions and reviews. Then, the most intense activity, which is the evaluation itself, must be organized. This fourth step requires on-site visits to conduct reviews and interviews.

The fifth step is reporting. At this stage, the results of visits and interviews are used to draw up conclusions that will need to be validated with the people involved. Professional judgement is required in order to establish a consensus on whether or not current practices comply with the standard or model. The fifth step is the last one common to both ISO 9001 and CMMI or ISO 15504 evaluations, and is the one where results are documented and communicated to the organization's management.

Finally, the sixth step in Figure 2.3 is an activity peculiar to process evaluation, that is, generating a graphical representation of the results. This representation is referred to as a *key areas profile*, which shows gaps by maturity level for each key area in the evaluation model.

Figure 2.4 provides an example of a typical process area profile. Each maturity model practice is color coded (green, yellow, orange, and red) following its assess-



**Figure 2.4** Example of a S<sup>3</sup>m® area profile.

ment. A color code is typically used to visually identify whether or not the practice fully, largely, or partially meets, or does not meet, its objectives. Figure 2.4 follows the ISO 15504 recommended rating categories of N, P, L, and F. The CMM ratings are slightly different: Yes, No, Doesn't apply, and Don't know. Finally, an ISO 9001 rating is also different: Satisfied or Not satisfied. In ISO9001 the decision is either you are conforming or not to a clause of the standard.

### 2.4.3 Evaluation Types

Many terms related to process evaluation are used in the literature. The term *appraisal* is used to describe a general diagnostic method without specifying its goal. The term *assessment* has a historical interpretation of an internal evaluation with improvement as its main goal. Finally, the term *evaluation* is associated with an external evaluation activity or the evaluation of a third-party entity.

The following evaluation types have been identified and described:

- *Audit*, the goal of which is to investigate a supplier's or service subcontractor's development process in order to make a business decision. Similarly, the SEI's capability evaluations (SCE) are also used to evaluate a software provider and to make a decision about its eligibility as a solution provider.
- *External evaluation*, the goal of which is to evaluate the development process with a team of personnel external to the organization (internal evaluation performed with the help of consultants).
- *Joint evaluation*, the goal of which is to evaluate the development process with a team of people from both organizations (typically for interfaces with subcontractors).
- *Self-evaluation* or *internal evaluation*, the goal of which is for an organization to evaluate its own development processes with people from its software development organization. “Quick evaluation (approximation) done by an organization, by and for itself, with continuous improvement as the goal” [SEI 2002].

CMMi categorizes different evaluation methods into three classes (see Figure 2.5). This classification recognizes, then, that many evaluation methods are available and that they possess identifiable characteristics.

Class A groups together complete evaluations such as SCAMPI [SEI 2000]. These evaluation methods are the most thorough, with a goal of building organizational consensus by proposing improvements and pointing out the strengths that can be leveraged. It is in this class that evaluations that conform to the international ISO standards are found. Class B describes smaller-scale methods, which are often called “miniassessment” or “preassessment” methods. These evaluations require smaller teams and less effort. They serve as substitutes for full evaluations and for monitoring progress between two full evaluations. Class C describes even lighter approaches, often called “microevaluations” or the “survey approach.” With this

Characteristic	Class A	Class B	Class C
Usage mode	Rigorous	Initial, partial and self-evaluation	Quick overview, incremental
Establishes a maturity level	Yes	No	No
Advantages	Detailed coverage	Partial coverage	Quick overview of a specific practice
Disadvantages	Requires many resources	No detailed coverage. Cannot be used to evaluate maturity	Few resources, less acceptance and low depth of study
Promoter	Senior manager	Promoting manager of an improvement program	Internal manager
Team size	4-10 people and an evaluation team leader	1-6 people and an evaluation team leader	1-2 people and an evaluation team leader
Team qualifications	Experienced	Average experience	Average experience
Prerequisites for evaluation team leader	Evaluator in chief	Evaluator in chief or a person trained in the method	A person trained in the method
Evaluation team composition	External and internal	External or internal	External or internal

**Figure 2.5** CMMi evaluation classes characteristics.

approach, it is only feasible to obtain a general (i.e., approximate) idea of an organizational practice's current state.

In summary, organizations have access to different methods, classes, and types of evaluation to evaluate their organizational practices according to their needs. CMMi recognizes this by allowing the development of a method tailored to the different goals chosen by the organization.

## 2.5 HOW IS A MATURITY MODEL DESIGNED?

Many software engineering maturity models have been proposed, but only a few authors have studied and documented the approach used to design such maturity models.

### 2.5.1 The Trillium Design Process

Trillium is the name given to the maturity model designed by Bell Canada and Noratel [Trillium 1994]. This maturity model focused solely on enhancing the capability

of software developed for telecommunications products and services. April [April 1995a, April 1995b] and Coallier [Coallier 1999] have described the process used to design the Trillium model. This process model was probably the first maturity model to move away from the staged representation, instead using a continuous representation that highlights the fact that a practice can be performed at all levels of maturity. The Trillium architecture influenced the CMMi and ISO 15504 architectures, both of which now offer a continuous representation. A four-step approach to the development of the model has been documented.

In Step 1, a workgroup composed of software engineering and telecommunications experts was formed, which identified the sources of relevant practices to build a domain-specific maturity model. The Trillium model design team used the CMM version 1.1 practices to get started. Using the CMM model as a starting point, its five-level maturity structure was used as the Trillium model's architecture.

Step 2 was a mapping between CMM and ISO 9001:1994 practices. CMM practices can be modified to accommodate this mapping. This is a consolidation of two practices with similar texts and the same intent. In this activity, certain ISO 9001:1994 practices were identified because they did not match anything in the CMM. Then, the Trillium model architecture was adjusted (creation of roadmaps or areas) to ensure that they appeared in the new model. At this point, the Trillium model possessed all CMM and ISO 9001:1994 practices. This mapping activity was then repeated for each additional standard to be included in the model. The following standards and exemplary industry practice guides were included in the design of the Trillium model:

- Bellcore standards
- Malcolm Baldrige National Quality Program
- IEC300 and IEEE standards

In Step 3, some Trillium-specific practices were added during each of the multiple model reviews to address concerns and additional areas that were important to the telecommunications industry. These additions were based on feedback from industry professionals and validated by a process of consensus building and multiple reviews. The designation of a given level is based on the general criteria found in Figure 2.6:

- First criterion. Practices considered “basic principles” and which contribute to a development project’s success are classified as level 2.
- Second criterion. Practices being considered for organization-wide deployment or fundamental to continuous software development process improvement are classified as level 3.
- Third criterion. Practices related to CASE tool technology or which characterize advanced processes or techniques (e.g., change management, failure prevention, statistical process control, and measurement) are classified as level 4.

	Level 1	Level 2	Level 3	Level 4	Level 5
Process	None	Project-oriented	Organization-wide	Organization-wide	Organization-wide
Technology		State of the art in the mid-1980s	State of the art in the start of the 1990s	State of the art in the mid-1990s	State of the art in the end of the 1990s
Standards	None	IEEE, SEI model level 2+, Bellcore TR-NWT-000179 (75 %)	IEEE, SEI model level 3, ISO 9001, CEI 300 (system)	SEI model level 4+, CEI 300 (software)	SEI model level 5
Process improvement	None	Unstructured	Deployed	Systematic	Systematic

**Figure 2.6** Trillium maturity levels.

- Fourth criterion. Level 5 practices refer to technical advances using automated tools, formal methods and the use of strategic information databases.

Finally, in Step 4 experimentation with the Trillium model at Nortel and Bell Canada allowed the model to be tested and many improved versions have been published.

### 2.5.2 ISO 15504 Design Process

Graydon [Graydon 1998] describes the ISO 15504 reference model's design process. The project started with a document describing the requirements, which was then reviewed by experts on the proprietary models. In order to come up with a description of area architecture and practices, the experts decided to include all software life-cycle process steps using the ISO 12207 standard, as well as the following maturity models: CMM, British Telecom Software Assessment Method (SAM), Bootstrap, and Trillium. This effort designed five process domains: (1) customer-supplier, (2) engineering, (3) project, (4) support, and (5) organization.

In the next design iteration, the work group had to find the key activities in each process in order to identify the base practices. Base practices are implemented first and typically appear at the first level of the maturity model. Once determined, the practices on this list were grouped into capability levels. Discussion among experts followed in order to determine groupings within six maturity levels (from zero to five) that could help organizations plan their process improvements through a recognizable rationale of progress across levels. The ISO 15504 work group formalized key concepts for each level; for example, the concept of institutionalization is used as a key characteristic for level 3. Generic practices that apply to all practices are found: (a) documentation, (b) follow-up, and (c) measurement.

### 2.5.3 CMMi for Services Design Process

Hollenbach describes the CMMi for services reference model's design process. The CMMi for services is based on the CMMi architecture, which was then reviewed by 12 experts, some of the models were mapped. The team proceeded with three design "build" stages with the following steps:

- Identified CMMI v1.1 development language in foundational model content
- Refined CMMI v1.1 definitions to ensure applicability to services
- Mapped IT service models to CMMI v1.2
- Identified new model content from model gap analysis
- Designed architectural structure
- Identified service process areas
- Designed practices within process areas
- Added service material to existing process areas
- Packaged and released draft for comment

### 2.5.4 Summary

In summary, a maturity model is designed through:

- A consensus of experts
- Studying the architecture of current models
- Multiple mappings to standards and other approved source documents
- The addition of specialized practices peculiar to a specific domain
- A rationale for the classification of each practice into a specific maturity level
- Experimentation for model adjustment and improvement

## 2.6 INITIAL VALIDATION OF A MATURITY MODEL

The validation of process models may be pre- or post-release. The post-release option is less satisfactory. This is because lessons learned can only be internalized by changes in the model, which, by virtue of occurring late in the development cycle, will be costly. [Wang and King 2000 p. 30]

Attention was given to the validation of maturity models in the early days of the CMM. The validation activities were concerned with evaluating the effectiveness of a given standard, method or tool, comparing its use versus not using it where the measure of effectiveness is cost and quality. The DESMET study [DESMET 1994] was used for the design of the ISO 15504 trial empirical studies [MacLennan 1998, Buchman 1998, El Emam 1998a], defining three types of empirical study: a formal experiment, a case study, and a survey. The following three examples of maturity

model validation have used case studies. A case study is a way of conducting a trial evaluation in a realistic situation, in a complex environment, over which the investigator has little control.

Wang and King identify some deficiencies in the validation of process models, specifically that process models lack validation at the organizational, process, and attribute levels. At the organizational level, there is a need to identify and model a complete set of fundamental process categories. “Almost all existing process models are empirical-and-descriptive models. These models lack rigorous and formal description of model structure” [Wang and King 2000].

### **2.6.1 IT Service CMM Model—Initial Validation Approach**

The postrelease validation approach of the Service Capability Model [Niessink 2000] was a questionnaire based on key practices that was developed and previously pilot tested. Two different approaches and questionnaires were used for Niessink’s two case studies. The first was a quick-scan evaluation using a questionnaire; the second was an on-site assessment using an assessment plan. In this experimentation, the questionnaire’s validity was not documented. The questionnaire approach has challenges—internal consistency of maturity questionnaires can cause problems and must be assessed [Fusaro et al. 1997]. In his first case study, Niessink used a questionnaire and also assessed a quick-scan evaluation method. The second case study had two general goals: (1) to support the organization in determining the quality of its service processes and indicate directions for improvement, and (2) to use and test the IT Service CMM in a process evaluation. Without additional criteria, this type of validation can be classified as a case study.

### **2.6.2 CM<sup>3®</sup> Model—Initial Validation Approach**

Kajko-Mattson [2001a] reports the post-release validation of the CM<sup>3®</sup> Problem Management Maturity Model using a mix of qualitative and quantitative evaluations across 17 case studies. The quantitative evaluation approach faces a number of challenges, as it requires:

- A well-defined hypothesis
- Response variables directly derivable from hypotheses
- A detailed definition of all treatments to be applied
- A design that identifies and mitigates confounding effects that may affect the impact of the treatment
- A detailed description of the conditions of the study
- Detailed definitions of the data to be collected and their method of collection that allow measurements of the response variables to be calculated
- Valid statistical analysis techniques

The evaluation results presented by Kajko-Mattsson [2001a, pp. 145–165] are in the form of ratios of how many organizations have a process described by the model (for example: “Fourteen out of 17 organizations have defined and documented their problem management processes”). The CM<sup>3®</sup> model validation approach can also be classified as case studies, but not quantitative evaluations. This type of validation approach provides anecdotal evidence rather than conclusive proof.

### **2.6.3 ISO 15504 Model—Initial Validation Approach**

The ISO 15504 postrelease validation trials were quickly faced with a large number of variables and the requirement for a larger sample, far bigger than planned. As a result, the project’s management team excluded the formal experiment validation approach [El Emam, 1998a].

Surveys were then classified as inappropriate, as it was deemed too difficult to test many hypotheses for which supporting evidence may be inconsistent. Inconsistencies in the ISO 15504 validation trials appeared when using a small number of documents and data derived from a small number of validation trials (35 evaluations were planned).

The ISO 15504 project team found the use of case studies more appropriate, given the environment in which software process improvement is undertaken.

### **2.6.4 CMMi for Services Model—Initial Validation Approach**

By now, we all understand that maturity models use a number of trials to ensure the quality of the model. After the CMMi for services draft reviews, a number of pilot assessments were planned. The CMMi for services project team proposed a small number of case studies before the release of the new model.

### **2.6.5 Maturity Model Validation, Conclusion**

The validation of process models is rarely prereleased. These models lack rigorous and formal description of model structure. This still holds for even the most recent models.

The CMMi for services, ISO 15504, Niessink, and Kajko-Mattsson studies all report initial postrelease maturity model validation activities that provide anecdotal evidence rather than conclusive proof. A number of case studies were used in all four cases.

## **2.7 WHAT IS THE TYPICAL ARCHITECTURE OF MATURITY MODELS?**

A model’s architecture refers to its fundamental structure and the relationship between maturity levels and sets of practices and process areas. Most maturity models

are based on the SEI's model architecture. The following models document upfront in their introduction that they are using the CMM's architecture.

### 2.7.1 CMMi Model Architecture

In 2002, the SEI introduced a revised four-level architecture with its CMMi model (See Figure 2.7). Before describing this architecture in detail, it must be noted that CMMi classifies the various model components into three categories: (1) required, (2) expected, and (3) informative.

At the top level, CMMi offers two representations of the same content to facilitate improvement activities. CMMi introduces a metastructure that oversees the different process areas. Either a continuous or a staged model can be chosen. Both representations are designed to offer essentially equivalent results. These two approaches were developed with the understanding that there can be different approaches to the measurement of process maturity.

A continuous representation is typically used for process improvement, while the staged representation is typically used for supplier evaluation. The two are slightly different and there is no published data assessing either one, so organizations should evaluate both before making a choice.

The continuous representation has the following characteristics:

- Allows selection of the improvement sequence that meets business goals and facilitates better risk management

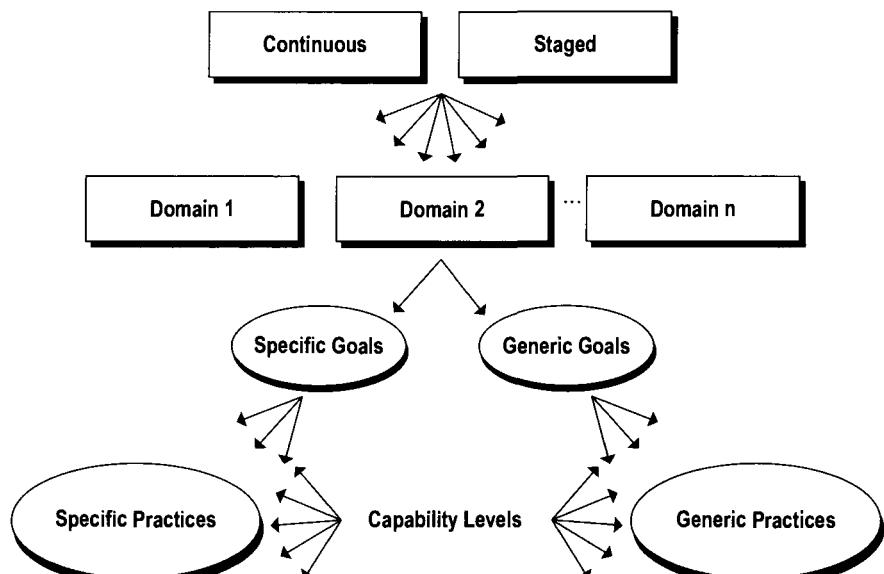


Figure 2.7 CMMi architecture [SEI 2002].

- Allows comparisons between organizations (by process area)
- Allows an easy migration from the interim EIA 731 standard of the Electronic Industries Alliance to CMMi
- Allows for easier<sup>1</sup> comparison with the International Organization for Standardisation and International Electrotechnical Commission's ISO 15504 reference model, its process area organization being similar

The staged representation has the following characteristics:

- A documented, ordered suite of improvements, starting with fundamental practices and progressing through successive levels, each serving as the foundation for reaching the next level's goals
- Internal comparisons and comparisons between organizations made possible using exactly the same staged maturity levels
- Easy migration from the CMM to CMMi
- An easily and widely recognizable single ranking summarizing the evaluation and comparisons between organizations.

The second level of the CMMi architecture is composed of process areas. A key process area is defined in section 2.2 as “a set of key practices that pertain to the same general domain of activities” [SEI 2002, s. 2]. The CMMi adds a goal to this definition: “These key practices, used collectively, must satisfy a series of goals considered important in achieving a significant improvement in this area” [SEI 2002, s. 2].

The third level of the CMMi architecture is composed of specific and generic goals in process areas. Specific and generic goals are mandatory and must be present, planned, and executed by the organization’s processes. The fourth level of CMMi architecture is composed of specific and generic practices. These specific and generic practices represent typical routine implementations that are expected to be used in organizations that develop and maintain software. These practices, as described in the CMMi model, or acceptable alternatives, must be present, planned, and executed by the organization’s processes in order to consider the goals as having been met. Generic practices have a different scope. They ensure organization-wide deployment, and they aid in developing an efficient, repeatable process implemented throughout the organization.

To allow a repeatable evaluation, different characteristics must be observable for each of the evaluation model’s practices, such as: (a) execution responsibilities, (b) execution capability, (c) deployment management, and (d) deployment verification. Level 1 practices are often called “base practices” and practices from levels 2 to 5 are often called “advanced practices.”

<sup>1</sup>There are, however, still difficulties in comparing CMMi and ISO 15504: different processes, process domain classification, taxonomy, and different rating criteria make comparisons difficult.

## 2.7.2 The ISO 15504 (SPICE) Reference Model

For an ISO-conformant software-process evaluation, the model's conformity with the ISO 15504-2 reference model must be verified. This ISO/IEC reference model describes five software process categories:

1. CUS, Customer-supplier
2. ENG, Engineering
3. SUP, Support
4. MAN, Management
5. ORG, Organization

In addition, capability evolution must be expressed in terms of these processes' attributes, grouped in six ISO recognized maturity levels: 0, unstructured; 1, executed; 2, managed; 3, established; 4, predictable; and 5, dynamic change.

The five process categories are grouped according to the software life-cycle dimensions defined in ISO 12207: the customer-supplier and engineering processes are part of the primary ISO 12207 life cycle, the support process is part of the ISO 12207 support life cycle, and the two "management" and "organization" processes are part of the ISO 12207 organizational life cycle.

Each maturity level has its own set of attributes for measurement purposes:

Level 0—No recognizable attribute.

Level 1—Process performance attributes.

Level 2—Process performance management attributes; Intermediate deliverables management attributes

Level 3—Process definition attributes; Attributes of resources associated with processes

Level 4—Process measurement attributes; Process control attributes

Level 5—Process change attributes; Continuous improvement attributes

At level zero, there is no evidence that process attributes are realized in a systematic and repeatable manner.

At level 1, process performance attributes under observation can be measured by looking at identifiable inputs and outputs. The process's objective and its inputs and outputs are known. Deliverables that support the process's goals are produced. Something is put in place, exists, or some activity is carried out within the organization. This can either be on a project-by-project or organizational-wide basis.

At level 2, level 1 processes are now managed (planned, executed, and verified) with defined goals. Process performance targets are defined (quality, delays, and resource use). Actors' responsibilities and authorities are defined and the process performance management is carried out according to the goals. There is a written procedure that explains what the specifications and expectations are as well as how and

by whom the process is controlled and executed. This procedure is understood and constantly used by the staff. Changes are managed and products verified for conformity.

At level 3, level 2 managed processes are now implemented according to a defined process, which is documented in conformity to software engineering principles. Processes are able to meet planned targets. Standardized process and adaptation guides for specific situations are available. Process performance monitoring is carried out according to a documented standardized process. Historical data is accumulated in order to improve existing processes. Experience with process use is analyzed for further improvements. The infrastructure, roles, responsibilities, and experience necessary to execute a process are identified and documented. Necessary resources are available, allocated and utilized to execute processes as documented.

At level 4, processes established at level 3 are executed within established control limits. This maturity level's attributes are the following: process and product measures are identified and accumulated for each goal, trend and performance measures are produced and analyzed, and processes are controlled with measurement-based analysis techniques.

At level 5, predictable level 4 processes become dynamically modifiable. This maturity level's attributes are the following:

- The impact of a change to existing processes is evaluated.
- Each change to be introduced is managed to minimize interruptions/impacts to the performance of processes in use.
- Continuous-improvement targets are defined for each process.
- Sources of current and potential problems are identified.
- Improvement opportunities are identified and an improvement implementation strategy is established and deployed throughout the organization.

ISO 15504 standardizes six descriptive process components:

1. *An identifier*, which identifies one of the five categories, followed by a unique sequentially numbered structure from the general to the specific (e.g., CUS 1: Acquisition process; CUS 1.2: Supplier selection process)
2. *A name*—a short sentence that encapsulates the process's objective
3. *A Process type* (from 1 to 5):
  - 1—Base process identical to the ISO 12207 process
  - 2—Extended process that adds to ISO 12207's existing processes
  - 3—New process that is not covered by ISO 12207
  - 4—Component that is an activity (or a group of activities) from an existing ISO 12207 process
  - 5—Component extension that adds to the activities (or a group of activities) of an existing ISO 12207 process with additional elements
4. *Process goal*—a descriptive text about the process's high-level goals

5. *Expected process results* to describe the observable results of a correct process implementation. Has to refer to a list of items that must appear immediately after “following a success in process implementation.”
6. *Note*—additional information about the process.

Capability-level implementation is done with the help of a set of attributes. These attributes are used to determine if a process has reached a certain capability level. Each attribute measures a specific aspect of process capacity. Each attribute is measured on a percentage scale. The standardized scale for evaluated attributes consists of the following irregular intervals:

- N—Not reached: 0–15%  
P—Partially reached: 15%–50%  
L—Mostly reached: 51%–85%  
F—Entirely reached: 85%–100%

## 2.8 AN INVENTORY OF SOFTWARE ENGINEERING MATURITY MODELS

Since the mid-1980s, both the industry and the software engineering research community have proposed a number of maturity models. The concept proved so appealing that maturity models were developed for sectors others than software such as human resources management, financial management, and primary care.

Sheard [Sheard 1997] summarized the best-known models in the United States and their relevant standards (Figure 2.8). Moore [Moore 1998] completes this list with Canadian and European models. A brief description of each of these models is presented in Figure 2.9.

Figure 2.9 includes the origin of the models and whether or not they are publicly available. Descriptions of models that have been replaced (a) by more recent ones, (b) by more popular ones, or (c) because of a lack of updates, or which have become obsolete are written in italics. Only a few models have benefited from significant investment, gained market acceptance, and survived. What about other software engineering maturity models documented in the literature over the past few years? A number of them have been proposed since version 1.0 of the CMM was published in 1993 (see Figure 2.10). A number of authors have highlighted this proliferation of models and pointed out that it has become difficult to understand which of the recent proposals has been validated.

A number of many specialized software engineering topics have also been addressed in the literature:

- Networking [Vetter 1999]
- Security [Hopkinson 1996, Menk 1996]
- Adaptation of software skills [Dove et al. 1996]

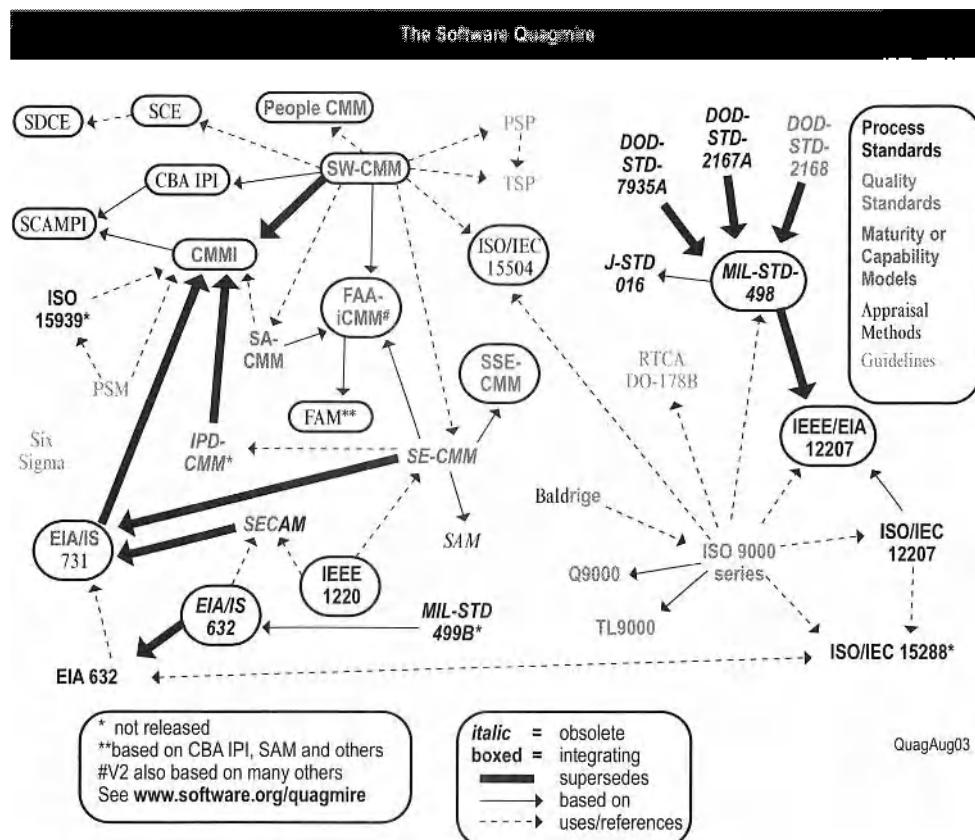


Figure 2.8 Framework quagmire [Sheard 1997].

- E-business and e-government [Tobia & Glynn 2001, Windley 2002]
- Ease of software use [Bevans 2000, Earthy 1998]
- Computer operations [Scheuing et al. 2001, Sribar 2001]
- Data management [Mullins 2002]
- Performance engineering [Schmiertendorf & Scholz 1999]
- Reuse [Topaloglu et al. 1998]
- Alignment of IS/IT and business [Luftman 2001]
- IS/IT management processes [IPWSM 1999]
- Specification management [Sommerville 1997]
- Business process reengineering [WITCHITA 1999]
- Software project management [Stratton 2000, Rayner 2001, Kerzner 2002, Crawford 2002, Schlichter 2002]

Model name	Origin and year	Public Access	Description
SW-CMM	SEI 1991	Yes	The first maturity model for software engineering, used for improvement as well as the assessment of software suppliers. From the book by W. Humphrey [Humphrey 1990].
TRILLIUM and CAMÉLIA	Bell Canada 1991	Yes	These models, created by Bell Canada and Nortel, integrate the SW-CMM, ISO9001 and other standards in order to evaluate development and maintenance capability for a telecommunications product. The last version was produced in 1996 (v. 3.0). CAMÉLIA, an improved version of TRILLIUM (in French only), was published in 1994.
Process Advisor	Pressman 1992	Yes	Pressman published a list of questions covering eight capability areas. This can be considered as a proposition for a simple model that compares an organisation's practices with the industry's best. [Pressman 1992]
Bootstrap	European ESPRIT projects 1993	No	The European software capability and process improvement model was developed as an ESPRIT #n5441 initiative. Bootstrap, ISO9001 and CMM ® were used as initial sources. The model was issued by the Bootstrap Institute in Brussels. Last version: 3.0 – 2000.
SQPA	HP 1993	No	Hewlett Packard model based on Capers Jones' work.
FAA-ICMM	FAA 1993	Yes	A model integrating the SW-CMM, the System Engineering CMM and the Software acquisition CMM, it was created to improve software at the U.S. Federal Aviation Administration.
STD	Compita 1994	No	Scottish Enterprise model accessible under license from Compita. This model complies with SPICE and uses a proprietary method called PPA.
SAM	BT 1994	No	British Telecommunications model for internal evaluation of its software processes.
SE-CMM	EPIC 1995	Yes	This is a model that describes essential practices for a systems engineering organisation.
People CMM	SEI 1995	Yes	This model describes key elements of human resources management and development in a software development and engineering organisation.
SSE-CMM	ISSEA 1996	Yes	Model (published in 1996) for security engineering processes.
IPD-CMM	EPIC 1996	Yes	This model for process improvement describes complete life cycle integration for all of an organisation's product development processes. This project stopped in 1997 and was never completed.
SA-CMM	SEI 1996	Yes	A model similar to the CMM for software in architecture, with an emphasis on the purchase of software solutions and their buyers.
SECAM	SEI	Yes	Model published in 1996 to support purchasing of software.
SECAM	INCOSE 1996	Yes	This model is based on a survey, with the evolution of current practices processes in systems engineering as its objective.
(SPICE) ISO15504	ISO/IEC 1998	Yes	ISO technical reports 15504 of software process improvement models and methods. A reference model is proposed and nine documents have been published.
EIA/IS 731	EPIC-INCOSE 1999	Yes	This model integrates the SE-CMM and the SECAM, resulting in the SECM, a model for systems engineering (EIA/IS 731).
CMMi	SEI 2002	Yes	Version 1.1 unifies many of the SEI's software maturity models. This new version renders many models obsolete: systems and software engineering, the IPPD, and software acquisition aspects.

Figure 2.9 History of main process models.

Year	Software Engineering Maturity Model proposals
1991	SEI 1991, TRILLIUM 1991, Koltun 1991
1992	Visconti 1992
1993	SEI 1993, BOOTSTRAP 1993, Kubicki 1993
1994	Camélia 1994, Krause 1994
1995	Curtis 1995, Zitouni 1995
1996	Burnstein 1996a, 1996b, Dove 1996, Hopkinson 1996, Menk 1996
1997	Sommerville 1997, Kwack 1997
1998	Topaloglu 1998, Bajers 1998, Earthy 1998, ESI 1998c
1999	Wichita 1999, Vetter 1999, FAA 1999, Garcia 1999, Schmietendorf 1999, IPWSM 1999, Niessink 1999
2000	Stratton 2000, Bevans 2000, COBIT 2000, Emmons 2000
2001	Kajko-Mattsson 2001a, 2001b, Rayner 2001, Scheuing 2001, Tobia 2001, Luftman 2001, Paydarfar 2001
2002	SEI 2002, Mullins 2002, Veenendaal 2002, Raffoul 2002, Kerzner 2002, Windley 2002, Schlichter 2002
2003	NASCIO 2003, USDOC 2003, Schekkerman 2003, Widdows 2003, Ream 2003, Duijnhouwer 2003

**Figure 2.10** Inventory of software maturity models (1991–2003).

- Software architecture [NASCIO 2003, USDOC 2003, Schekkerman 2003]
- Open-source software maturity [Duijnhouwer 2003]

This literature search has not revealed any comprehensive diagnostic techniques for evaluating the quality of the maintenance process and its unique activities, as described in previous sections. Our research team has looked at each of these models to identify contributions that could assist maintainers. Of the models inventoried in this list, only a handful include documented maintenance practices, sometimes accompanied by a rationale and references. However, none of them covers the entire set of unique maintenance activities of section 1.5. It is remarkable that, among these multiple additional models, only three specifically address software maintenance [Zitouni 1995, Niessink 1999, Kajko-Mattsson 2001].

We also found software maintenance practices in documents that are not considered maturity models per se. The Malcolm Baldrige National Quality Program, COBIT, and ITIL have been found to contain some valuable information for software maintainers and have been added to our list of documents to be mapped in detail.

## 2.9 SUMMARY

This chapter presented an overview of the concepts of process improvement based on maturity models. It reflects a specific domain with its specific terminology and history. First, it confirmed that CMMi does not cover software maintenance adequately because of its primary focus on software projects using a project manage-

ment approach. It also showed that CMMi does not address the specific processes and activities of software maintenance.

The literature survey of software engineering maturity models has not revealed any current proposal that addresses maintenance process and unique activities, as described in the previous sections. None of them covers the entire set of topics and concepts of the process model introduced in the previous chapters.

The differences between ISO 9001 and a maturity model were also explored, confirming that organizations tend to see them as complementary to one another. Evaluation methods and the different ways in which they can be used were also described. More important was the study of the architecture, development steps, and validation approaches of such models, describing how a maturity model is built by a consensus of experts mapping and assigning an inventory of domain-specific practices to each maturity level. As well, most maturity model designers choose to follow the CMM in their architecture, structure, and number of maturity levels.

A validation approach, using case studies, was also used by ISO 15504, as well as the IT Service CMM, the CM<sup>3®</sup> maturity models, and the CMMi for services. It is currently the preferred and most widely used validation approach for a new software engineering maturity model.

Last, only a few documents contain exemplary practices aimed at the software maintainer.

## 2.10 EXERCISES

1. What are the characteristics of an immature maintenance group? Describe what must change in order for the group to climb up the maturity levels.
2. Describe how Feiler process steps have influenced SEI work and their well-known CMMi model.
3. Explain what is meant by software capability. Does this concept also apply to software maintenance?
4. You are the employee of the month. Each employee of the month has his picture on the company website and is given the additional assignment of dressing up as the company mascot for a day or performing a management assignment for the chairman of the board. You are lucky—you pick a management task from the employee of the month hat. The company has decided to obtain an ISO 9001:2000 certification. However, the software development department has a CMMi certification. At the previous board meeting, everyone was confused about the various certifications, and someone asked your boss to figure out, for the next board meeting, the level of duplication of effort that exists in these certification schemes. The chairman of the board is a busy man and never wants to take more than 5 minutes on these topics. Your boss delegates this issue to you. You must thus explain (in less than 5 minutes) the context of use of these two approaches and their complementarities.
5. You return to the office after taking a graduate course on software mainte-

nance. You have a copy of the capability evaluation model for software maintenance, the S3<sup>m®</sup>. It is spring, and your boss has gone off to play golf with a customer. He left you with the responsibility of choosing an evaluation method for this maturity model. He told you that you will have to work with Jeff, who currently carries out ISO 9001:2000 audits and knows this subject inside out. How do you prepare, before meeting Jeff, to ensure that your project does not go off the rails?

6. Describe the typical architecture of a process maturity model. Comment on the approaches used for the following maturity models: (a) Capability Maturity Model for Operations [Sribar 2001] and (b) Outsourcing Maturity Model [Raffoul 2002].
7. List the relevant reference models useful in software maintenance. Explain why you would exclude some reference models.
8. The reference models provide additional information on the maintenance processes and activities listed in the previous chapter. Describe five activities that did not appear in that chapter. Include the source of your information and explain why, in your opinion, these activities were not included there.
9. You must make improvements within your maintenance group; Bill tells you that he heard about something similar somewhere else. You must:
  - (a) Use standards for problem monitoring and taking corrective action
  - (b) Identify measures for software quality
  - (c) Increase the satisfaction level of contributorsFind, with Bill's help, information about a similar improvement project. Focus on elements that can help you with process improvements.

# Chapter 3

---

## Foundations of the S3<sup>m®</sup> Process Model

### This chapter covers:

- The organizational context of software maintenance
- S3<sup>m®</sup> classification of software maintenance processes based on the structure of ISO 12207:
  - Primary processes
  - Support processes
  - Organizational processes
- S3<sup>m®</sup> maintenance process domains and key process areas

### 3.1 INTRODUCTION

This chapter presents the Software Maintenance Maturity Model, S3<sup>m®</sup>, a model that presents in a structured way, organized by maturity levels, a large number of exemplary practices and their references.

Poor knowledge of exemplary practices and their benefits to software maintenance activities can lead to inefficient or inappropriate investment. This, in turn, (a) creates dissatisfaction on the client's part, (b) hinders optimization of delays and service costs, and (c) delays the deployment of improvements, as well as innovation and investment, in information technology.

Organizations can use the S3<sup>m®</sup> model to launch and sustain a continuous improvement program tailored to software maintenance by initially benchmarking their current maintenance practices against this model. This will help maintenance organizations identify their strengths and weaknesses in delivering software maintenance services. With the identification of a gap, maintenance organizations can identify, through comparison with the model, what issues to address and how to address them, and by doing so improve their software maintenance processes.

The scope of a model such as S3<sup>m®</sup> is to deal with the software maintenance processes that are under an organization's direct control. A practical approach is used to apply proven knowledge in software maintenance engineering to offer relevant practices to improve software maintenance, and to do so for all types of industries and for organizations of any size.

It is an organization's responsibility to use this model for meeting their business objectives while taking into consideration the organization's costs and constraints.

The scope of the S3<sup>m®</sup> model is restricted to small maintenance activities, and this process model is not appropriate for software maintenance projects of larger scope, which should be managed as projects using project management techniques. For the maintenance workload requiring project management expertise and techniques, CMMI and other maturity models should be used.

#### The S3<sup>m®</sup> model

- Is based on the customers' perspective
- Is relevant for the maintenance of application software (a) developed and maintained in-house, (b) configured and maintained in-house or with a subcontractor's help, and (c) outsourced to an outside supplier
- Provides references and details for each exemplary practice
- Offers an improvement approach based on road maps and maintenance categories
- Covers ISO 12207 and ISO 14764 software life-cycle processes and maintenance standards
- Covers most ISO 90003:2004 characteristics and practices
- Covers relevant parts of the CMMI, a reference model for software improvement
- Conforms to the ISO 15504 international standard and covers those of the model's practices that are relevant to software maintenance (ISO 15504 part 2)

The S3<sup>m®</sup> model integrates references to additional software maintenance practices documented in other software and quality improvement models:

- ITIL Exemplary Practice Version 3 [Central Computer & Telecommunications Agency 2007a, 2007b, 2007c, 2007d, and 2007e];
- IT Service CMM [Niessink et al. 2005];
- Corrective Maintenance Maturity Model (Cm<sup>3</sup>) [Kajko-Mattsson 2001a];
- Maintainer's Education and Training Maturity Model (Cm<sup>3</sup>) [Kajko-Mattsson et al. 2001b];
- COBIT® Version 4.1 [IT Governance Institute 2007];
- Malcolm-Baldridge [Malcolm Baldridge National Quality Program 2007];

- Camelia [Camelia 1994]; and
- Zitouni and Abran [Zitouni 1996].

## 3.2 CONTEXT OF SOFTWARE MAINTENANCE

It is important to understand the scope of maintenance activities and the context in which software maintainers work on a daily basis (see Figure 3.1). There are indeed multiple interfaces in a typical software maintenance organizational context:

- Customers and users of a software maintenance interface (labeled 1)
- Upfront maintenance and help desk interface (labeled 2)
- Computer operations department interface (labeled 3)
- Developer interface (labeled 4)
- Supplier interface (labeled 5)

Taking into account these interfaces, which require daily service, the maintenance manager must keep the application software running smoothly, react quickly to restore service when there are production problems, meet or exceed the agreed-upon level of service, and keep the user community confident that they have a dedicated and competent support team at their disposal that is acting within the agreed-upon budget. Key characteristics in the nature and handling of small maintenance requests have been highlighted in [Abran 1993a], for example:

- MRs come in on an irregular basis and cannot be accounted for individually in the annual budget-planning process.

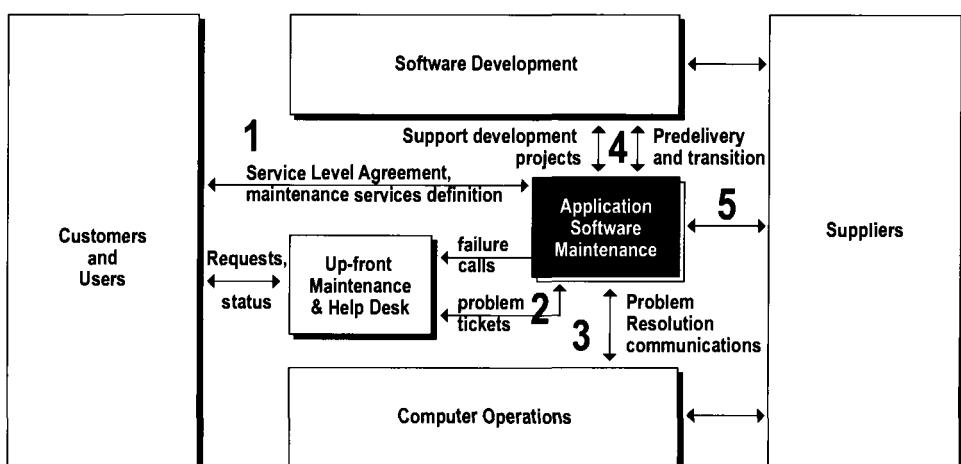


Figure 3.1 Software maintainers' context diagram.

- MRs are reviewed and assigned priorities, often at the operational level; most do not require senior management involvement.
- The maintenance workload is not managed using project management techniques, but rather queue management techniques.
- The size and complexity of each small maintenance request are such that it can usually be handled by one or two resources.
- The maintenance workload is user services oriented and application responsibility oriented.
- Priorities can be shifted around at any time, and MRs for application software correction can take priority over other work in progress.

A first interface is located between customers/users and maintainers. The customer interface operates between the managers of each organization. It consists typically of defining and managing the software maintenance services that are offered. It conducts negotiations about service scope, objectives and priorities, budgeting/pricing, and user-satisfaction-related activities. It is documented in a contractual (or internal) agreement, which is an SLA [April 2001].

Once this agreement is in place, the users can have access to support services from the software maintenance engineer on a daily basis. The interface with users differs from the customer interface, as it is more operational in nature. This interface (labeled number 2 in Figure 3.1) can be located in many different organizations according to different organizational models. The help desk service has sometimes been found to be part of the maintenance organization, sometimes to be part of the operations organization, and sometimes located in an independent product support organization (up-front maintenance organizations [Khan 2005, pp. 80–81]).

What is a standard practice is the provision by IS/IT organizations of an offer of a centralized help facility to their users. The problem-reporting activities, conducted through this interface, have been well documented as part of a problem management taxonomy published by Kajko-Mattsson [2001a].

To be effective, a mechanized problem reporting process is used that ensures efficient communications for quick resolution of failures and other support requests. A specific user request, sometimes called a “ticket,” will typically circulate between the help desk, software maintenance, and operations in order to isolate a problem.

A third important maintenance interface deals with the computer operations organization [Central Computer & Telecommunications Agency 2007d]. Computer operations personnel are the custodians of the infrastructure supporting the application software. They typically handle all the operations, support, and maintenance issues associated with the workstations, networks, and platforms. They conduct activities like backups, recovery, and systems administration. The user is rarely aware of, or involved in, internal exchange of information between software maintainers and operations. This interface also includes less frequent activities such as coordination of service recovery after failures or disasters in or-

der to help restore access to services, within agreed-upon SLA terms and conditions.

The fourth interface describes the relationship between software developers and maintainers. Some developers retain the maintenance of the software they develop. In this case, the interface between software development and software maintenance does not exist. Many organizations choose to have separate organizations for development and maintenance. When this interface exists, the root cause of some maintenance problems can usually be traced to development, and it is recognized that the maintainers need to be involved and exercise some form of control during predelivery and transition of the software from development to maintenance. Other services are exchanged across this interface. Many contributions are made by maintainers to concurrently support, and sometimes be involved in, a number of large development projects. The maintainer's knowledge of the software and data portfolios is of great value to the developers, who need to replace or interface with legacy software. For example, some of the key activities of the maintenance engineer would be (a) development of transition strategies to replace existing software, (b) design of temporary or new interfaces to legacy software, (c) verify business rules or assist in understanding the data of existing legacy software, and (d) assist in data migration and cutover of new software.

The last interface (labeled 5 in Figure 3.1) addresses relationships with a growing number of suppliers, outsourcers, and enterprise resource planning (ERP) vendors. The maintainers have a number of different relationships with suppliers. For example:

- With subcontractors who are part of the maintenance team and provide specific expertise and additional manpower during peak periods
- With suppliers who have yearly maintenance contracts providing specific support services for their already-licensed software (i.e., ERP and proprietary software)
- With maintenance outsourcers who might replace, partially or completely, a software maintenance organization for some software of the IS/IT portfolio

To ensure good service to users, software maintainers must develop a good understanding of the many contract types and manage them efficiently to ensure supplier performance, which often impacts service quality.

### **3.3 PROPOSED CLASSIFICATION OF SOFTWARE MAINTENANCE PROCESSES**

Wang and King [2000] suggested that a set of complete process categories be identified and modeled. This model can be formally defined and will ease the prevalidation activities. In order to develop a high-level process model for software maintenance, a list of processes and activities was inventoried. It was realized early in this project that the level difficulty in building a process model that would include the

current international standard content, as well as the unique activities presented by the SWEBOK software maintenance chapter, was high. It was also important to present a process model with which maintainers in the industry would associate. To achieve this goal, the software maintenance processes are grouped into three classes (as shown in Figure 3.2). The main idea is to provide a representation similar to that used by the ISO 12207 standard, but with a focus on software maintenance processes and activities:

- Primary processes (software maintenance operational processes) are processes that belong to a technical organization.
- Support processes (supporting the primary processes) are processes that belong to a supporting organization.
- Organizational processes offered “by a top-level administrative” [Wang & King 2000] are processes that belong to organizations like information systems (IS) or other departments of the organization (for example, training, finance, human resources, purchasing, etc.).

This generic software maintenance process model helps explain and represent the various key software maintenance processes.

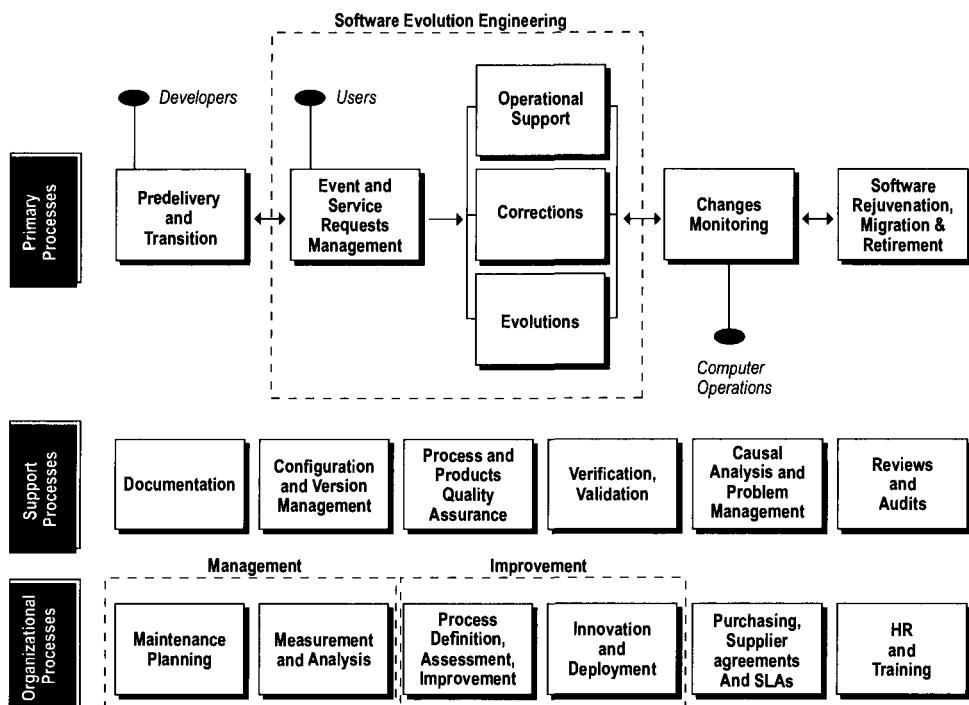


Figure 3.2 A classification of the software maintainer's key processes.

### 3.3.1 Software Maintenance Operational Processes

The operational processes (also called primary processes) that a software maintenance organization uses are initiated at the start of software project development, beginning with the *predelivery and transition process*. Transition is not limited, as is the case with some standards, to the moment when developers hand over the system to maintainers, but rather ensures that the software project is controlled and that a structured and coordinated approach is used to transfer the software to the maintainer.

In this process, the maintainer will focus on the maintainability of this new software, and the process implies implementation to support the developer during the system development life cycle.

Once the software has become the responsibility of the maintainer, the *event and service request management process* handles all the daily issues, PRs, MRs, and support requests. These are the daily services that must be managed efficiently. The first step in this process is to assess whether a request is to be addressed, rerouted, or rejected (on the basis of the SLA and the nature of the request and its size) [April 2001].

Accepted requests are documented and prioritized. They are then assigned to one of the following service categories:

1. Operational support process (requests that do not necessitate any modification to software)
2. Corrections process
3. Evolutions process (which treats adaptive, perfective, and preventive requests)

They are then processed. Note that certain service requests do not lead to any modification of the software. In the model, these are referred to as “operational support” activities. In S3<sup>m®</sup>, operational support activities consist of:

- Production of a one-time extract of information from existing systems
- Provision of technical support, consulting, monitoring, and problem management to developers and operations
- Helping customers and users to better understand the software, a transaction, its data, or its documentation

The next primary process is the *change monitoring* process, which ensures that the operational environment has not been degraded by the promotion of a change in production. Maintainers must always monitor the behavior of the application software and its environment for signs of degradation, especially after a change. They will quickly warn other support groups (operators, technical support, scheduling, networks, and desktop support) when something unusual happens and judge whether or not an instance of service degradation has occurred that needs to be investigated. The last primary process addresses three types of maintenance activities:

(1) rejuvenation activities to improve maintainability, (2) migration activities to move software to another technical environment, and (3) retirement activities when software is decommissioned.

### **3.3.2 Software Maintenance Support Processes**

As in ISO 12207, a process that is used, when required, by an operational process is said to be an operational support process. In most IS/IT organizations, developers, maintainers, and operations share the responsibility for these processes. Support processes typically include:

- A documentation process
- A software configuration and version management process and tools
- A process and product quality assurance process
- The verification and validation processes
- The problem resolution process, which is often shared with operations
- The reviews and audits processes

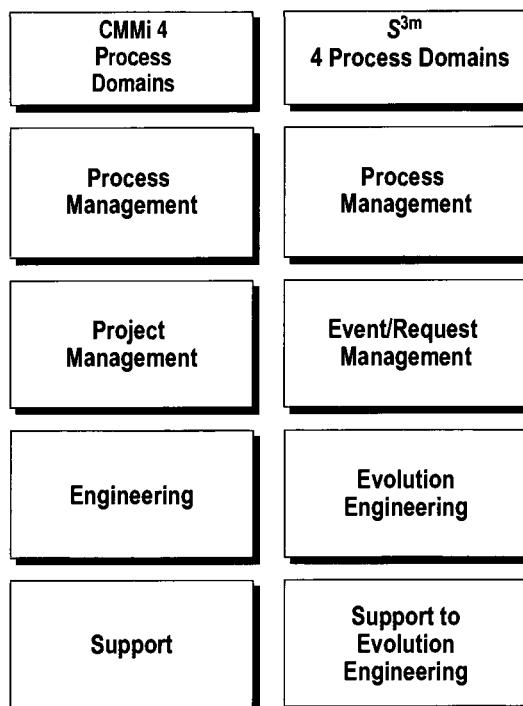
Some problems can recur or disappear before they can be identified. A causal analysis process that is used to identify long-standing issues and their causes is sometimes also found and classified as a support process. These are all processes that support software maintenance operational processes.

### **3.3.3 Software Maintenance Organizational Processes**

Organizational processes are typically offered by other departments in the organization (e.g., business planning, measurement, process-related activities, innovation, training, and human resources). Although it is important to measure and assess these processes, it is more important for the maintainer to define and optimize the operational processes first. The operational support processes, and then the organizational processes, follow these.

## **3.4 IDENTIFICATION OF PROCESS DOMAINS AND KEY PROCESS AREAS IN SOFTWARE MAINTENANCE**

The process model in Figure 3.2 helps in defining the process domains of a proposed software maintenance maturity model. Figure 3.3 presents, on the left, the four current CMMi process domains and, on the right, the proposed four process domains of software maintenance. This change is introduced to reflect software maintenance event/request management in contrast with project management used in software development. The proposed maturity model focuses on maintenance activities that are based on small maintenance requests, not large projects.



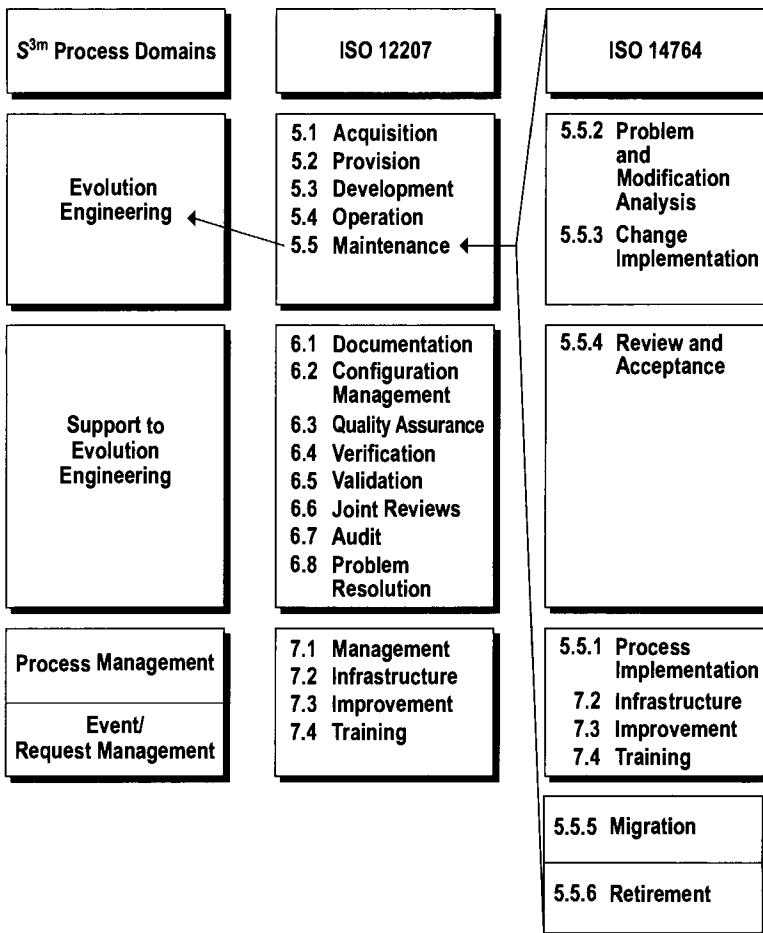
**Figure 3.3** Process domains: CMMI and S3<sup>m</sup>® software maintenance.

In software maintenance, it is also specified that engineering and support activities be centered around software evolution and not on its initial conception. To reflect this fundamental difference, the CMMI engineering domain was renamed “evolution engineering.” Evolution engineering is mostly concerned with modifying, rejuvenating, and migrating production software.

It has been mentioned in previous chapters that international standards, like ISO 12207 and ISO 14764, are now applicable to software maintenance processes and are part of that domain’s fundamental knowledge. These standards propose process models that are descriptive or prescriptive representations of activities that need to be followed or adapted by maintainers. It is then important to ensure that the S3<sup>m</sup>® model aligns its architecture to include a representation of all processes and activities identified in ISO 12207 [ISO 1995] and ISO 14764 [ISO 2006].

Figure 3.4 illustrates what each S3<sup>m</sup>® process domain must contain in order to cover these two international software standards. A maintenance model that conforms to these two standards should cover four subjects in its engineering process: (1) acquisition, (2) provision, (3) development, and (4) operations. Processes from ISO 14764 must also be comprehensively identified and included in the S3<sup>m</sup>® model.

This mapping helps identify key process areas (KPAs) of the proposed model. Figure 3.5 presents the proposed S3<sup>m</sup>® software maturity maintenance model’s four



**Figure 3.4** Standards topics of to be addressed by the S3<sup>m</sup><sup>®</sup> model.

process domains and 18 proposed KPAs. The following text explains the architectural decisions taken while designing the KPAs of the proposed model.

First, the CMMI process management area seems relevant for software maintenance. What needs to be done here is to insert a maintainer's view of those topics. The naming convention of the CMMI's five KPAs is modified slightly: (1) maintenance process focus, (2) maintenance process/service definition, (3) maintenance training, (4) maintenance process performance, and (5) maintenance innovation and deployment [Tornatzky 1990]. Only Zitouni's model proposed an additional KPA: "maintenance function organization," which can be integrated with one of the proposed KPAs [Zitouni 1995].

Decisions also need to be made concerning KPAs of the maintenance event/request management process domain. This is where the content of the model departs

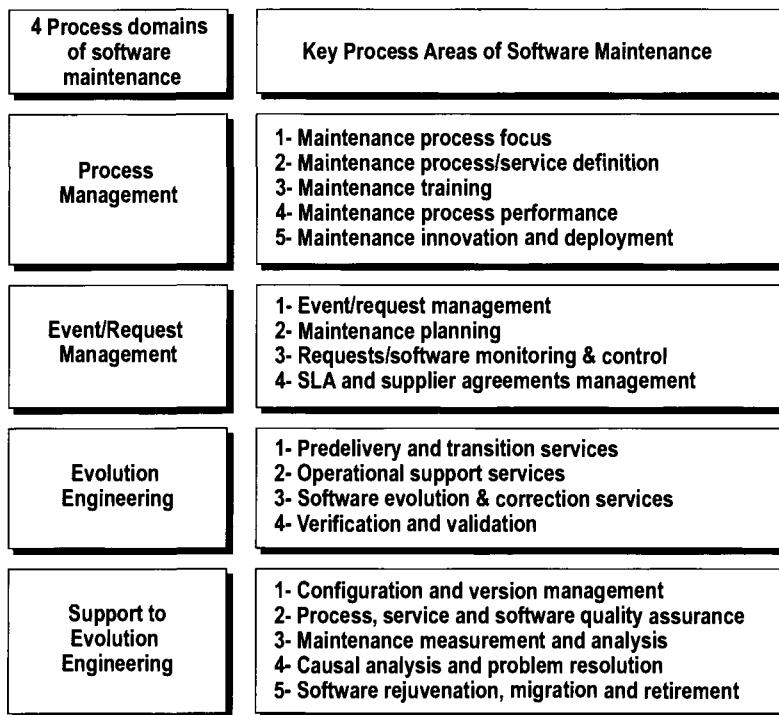


Figure 3.5 Proposed S3<sup>m®</sup> key process areas.

drastically from CMMi content. In the project management area, CMMi presents eight KPAs. The CMMi KPAs are: (1) project planning, (2) project follow-up and supervision, (3) supplier agreement management, (4) integrated project management, (5) risk management, (6) integrated teams, (7) integrated supplier management, and (8) quantitative project management. As indicated in Chapter 1 (refer to Sections 1.5 and 1.7), there are major differences between project management and maintenance event/request management.

The following decisions have been taken concerning the KPAs of this second process domain:

- Project planning has become “event/request management.”
- Planning, follow-up, and supervision aspects are being kept and content will be adapted to the software maintenance particularities.

Management of the service agreement, contracts, subcontractors and outsourcing are grouped under the title “SLA and supplier agreements management,” which reflects the importance of the SLA to maintainers.

Decisions concerning evolution engineering’s KPAs must also be taken. CMMi presents six KPAs: (1) requirement definition, (2) requirement management, (3)

technical solution, (4) product integration, (5) verification, and (6) validation. Here, it is of interest to keep as much equivalence as possible, because IS/IT organizations share processes and technologies across developers and maintainers. Improvement managers will also use CMMi to assess development and would like to have commonalities to ensure that there are as few differences in terminology as possible for similar maintenance activities. The following decisions have been taken:

- Maintainers deal with transition issues that are unique. A KPA has been created for this purpose.
- All services that do not translate into a change to software are grouped under a KPA named “operational support services.”
- Instead of project requirements, design, and construction, it was decided that a KPA would be created for all software modifications. In the “evolution” KPA, the following are included: adaptive, perfective, and preventive requests. Furthermore, corrections are identified as a different type of service that requires a different treatment in the model.
- Maintenance verification and validation move from an ISO 14764 support process to an engineering process, in order to align it with the CMMi.

Finally, a number of decisions concerning the KPAs of the “evolution engineering support” domain have been taken. The CMMi presents six KPAs: (1) configuration management, (2) quality assurance, (3) measurement and analysis, (4) integration into the organizational environment, (5) decision analysis and resolution, and (6) root cause analysis and resolution. Here, too, it is important to keep as much equivalence as possible for the same reasons as for the previous process domain. The following decisions have been taken:

- In addition to configuration management, maintainers deal with version management. A KPA has been created for this purpose.
- Some quality assurance activities of developers also apply to maintenance engineers.
- Measurement and analysis has also been kept. Here, both developers and maintainers will often have the same measurement repository to cover all IS/IT processes.
- Causal analysis and problem resolution concerns maintainers, as they must interface with many organizations to locate and fix failures. Long-standing and repetitive failures will be subject to root cause analysis.
- A KPA is defined to group together all rejuvenation activities (i.e., redocumentation, restructuring, reverse engineering, and reengineering). Again, in the proposed model, small-sized activities that do not require a project structure are addressed. This KPA also includes the ISO 14764 migration and retirement activities.

### 3.5 SUMMARY

In this chapter, the first part of the S3<sup>m®</sup> was introduced. It described the maturity model overview, organizational scope, and intended benefits. This was followed by explanation of how it should be implemented and the many references that it integrates. Also provided was an overview of the architecture and its maturity scales and provided examples, for each maturity level, of what is intended to be addressed by each level of maturity. Another intention of this chapter was to help the novice reader to better understand the allocation of a practice to a specific level of maturity and what is typically needed to move to the next level.

The next chapter presents part 2 of the proposed maturity model by describing the first process domain of the model, the *process management domain*, which contains five KPAs.

### 3.6 EXERCISES

1. You are working on the development of an evaluation plan when Marc tells you that it is sufficient to send out a letter to all employees and that little needs to be done to prepare for the evaluation. You secretly wish that Didier be assigned to this evaluation project! What is recommended in the literature with respect to preparing for an evaluation?
2. Name six success factors in software development and maintenance process improvement.
3. Tim has worked out his evaluation budget for next year, and he is looking at a time frame of 3 to 5 working days. He is also planning four evaluations within the coming year. What is wrong with his budget?
4. After your presentation to the board of directors, the chairman phones you. His brother-in-law is a consultant and has expressed his doubts to him about self-improvement. If software maintenance could implement self-improvement measures, the organization would not be in such a mess! You promised the chairman you would get back to him today with an e-mail. What arguments can you give him to justify going ahead with the improvement project to be carried out almost entirely in-house, rather than incurring the high fees of an external consultant?
5. Marc complained to the maintenance general manager because you chose Didier as the lead appraiser for the improvement project. You must explain your decision to Marc, and you have only 10 minutes before the last train leaves downtown for the suburbs. (You promised your wife that you would take your boys to hockey this evening. There is no room for discussion here!) How can you justify your decision and present your arguments quickly?
6. What are the four greatest objections to improvement in general? Describe approaches to refute them.

7. You have to convince your colleagues, who are very concerned about potential misuse of the evaluation results, that the administration will not use them to identify scapegoats. Several programmers, who are quite skeptical about this evaluation exercise, claim that it discourages innovation and that this model devalues their work. What arguments will you use to convince them that this evaluation exercise is sound?
8. After you have tabled your report to the board of directors, the accountant notices that there were no financial ratios in your report. His main criticism is that this exercise is a black hole into which thousands of dollars could be thrown without deriving any benefit from it. You are ready for the fight, since Jim has already discussed this with you. What arguments will you use to justify these maintenance improvement expenses?
9. Identify the key process areas within the S3<sup>m®</sup> that reuse the CMMi practices and those with practices specific to software maintenance.
10. A manager approaches you and indicates that he has found a process improvement model that can be used right away and that can solve all maintenance problems at a stroke. Dampen his enthusiasm!
11. Why does the S3<sup>m®</sup> refer so often to the CMMi and to all kinds of other documents?
12. Norman tells you that if the maintenance group does not implement the suggested practices exactly as recommended, this will lead to nonconformity! Mitigate his position.
13. Your boss has decided to adopt the process classification as described in the generic model. You looked at it, and you think that there is a problem. Your boss insists that it is you who must modify your approach. What will be your position and your argument?
14. Didier wishes to look over the model's reference documents in order to gain a better understanding of a practice. Roger tells him that this is not necessary and that it is more important to work with it in the model instead. What do you think?
15. A marketing rep has just spoken with an important customer and told him that you were S3<sup>m®</sup> level 3 qualified, and that this means both SEI level 3 and ISO 9000 certification. What clarification is required?
16. Explain the theoretical foundations of a road map.
17. What are the criteria required to ensure that a model is in conformity with the ISO 15504 standard?
18. Describe, in simple terms, the difference between levels 0, 1, and 2 of the model.

# Chapter 4

---

## Process Management Domain

### This chapter covers:

- An overview of the S3<sup>m®</sup> first process domain: process management
- For each of its key process areas, the objectives, goals, expected results, and links with other KPAs: process focus, maintenance training, process performance, and maintenance innovation and deployment

This chapter presents the overview and goals of the first process domain of the proposed model. The detailed practices for the zero, first, and second maturity levels of this process domain are presented in Chapter 8.

### 4.1 OVERVIEW

This first process domain focuses on the process management of software maintenance. An adequate management system for software maintenance processes must satisfy all the criteria of customer services and the technical domain.

This process domain recognizes the importance of human resources at the core of the software maintenance activities, through their daily contacts with customers and their execution of the many processes (see Figure 4.1). Software maintenance engineers use the processes, techniques, and tools at their disposal to satisfy their customers and users. Maintenance employees are well positioned to help in process definition, acquisition, and improvement.

This process domain covers the following five KPAs:

- Pro1: Maintenance Process Focus
- Pro2: Maintenance Process/Service Definition
- Pro3: Maintenance Training
- Pro4: Maintenance Process Performance
- Pro5: Maintenance Innovation and Deployment

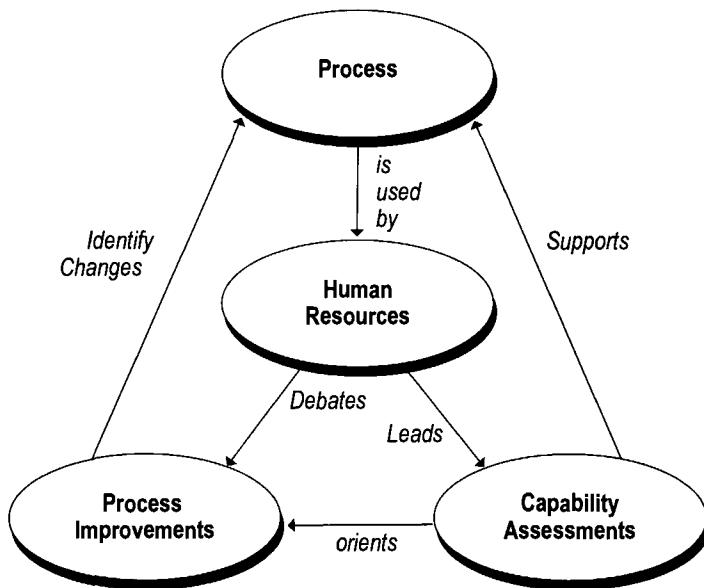
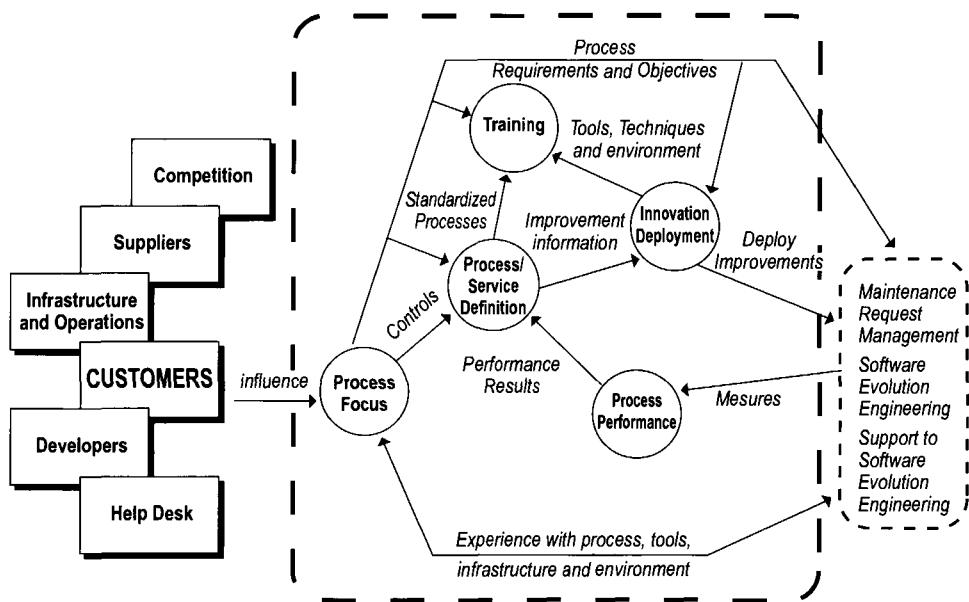


Figure 4.1 Process management context.

Figure 4.2 was inspired by a CMMi representation and can be used also to describe the software maintenance process context. It shows the interactions of the various KPAs of this first process domain with the other process domains of the model (represented outside the scope on the right-hand side).

The *Maintenance Process Focus* KPA helps the maintenance organization plan continuous improvements by gathering information from (a) customers, (b) its personnel, (c) its personnel needs of additional knowledge and competencies, (d) the current performance of maintenance processes, (e) the strengths/weaknesses of current maintenance techniques and tools, (f) the overall maintenance working environment, and, finally, (g) the feedback from the many software maintenance interfaces (as described earlier in Figure 3.1). All this information contributes to the *Maintenance Process/Service Definition* KPA, in which maintenance processes, techniques, and tools are assessed and reviewed to improve the performance of the maintenance engineer's daily tasks.

The *Maintenance Training* KPA identifies strategic needs for education and training, while focusing on the processes and also on the technical aspects. Training is developed internally or acquired by vendors/consultants with the objective of improving the competencies and knowledge needed for executing the maintenance processes. This proposed maturity model refers to the software maintenance training exemplary practices documented by Kajko-Mattsson [2001a] (which includes a maturity model for education and training for corrective maintenance).



**Figure 4.2** Process management KPA interactions, adapted from [SEI 2002].

The *Maintenance Process Performance* KPA establishes quantitative goals for (a) the quality and performance levels of the execution of maintenance processes, (b) the software products in operation, and (c) the intermediate products (artifacts) of software maintenance. Through this KPA, the maintenance organization measures how it achieves its business goals. The maintenance organization ensures, in coordination with the other IS/IT organizational units, the implementation of (a) the goals of the process/product measures, (b) the definitions of the process/product measures, (c) the reference points (baselines) for these measures, (d) a repository, and (e) the use of the measures (i.e., in estimation/performance prediction models). The software maintenance organization analyzes the execution data of its key processes in order to develop quantitative knowledge. This helps in assessing the quality of its deliverables and services, the performance of its processes, and the technologies currently in use.

The last KPA of this first domain is the *Maintenance Innovation and Deployment* KPA, which groups together practices to select and deploy innovations and improvement projects [Rogers 1995, Nord 1987]. These projects aim to improve the ability of the maintenance organization to meet its SLA goals for quality and process performance. The introduction of new techniques, tools, and procedures should be considered as implementing a new technology for an organization (that is, a technology innovation for such an organization). Decisions on technological changes must be based on facts using data and cost/benefit studies, as well as conducting experiments and controlled deployments.

## 4.2 MAINTENANCE PROCESS FOCUS KPA

The Maintenance Process Focus KPA considers the definition, maintenance, improvement, and deployment of the software maintenance organizational processes. It also ensures the coordination of the activities required for its promotion, evaluation, and improvement.

The objective of this first KPA is to ensure that the responsibility exists to collect information on needed improvements, develop the insights needed to analyze them, and identify improvement solutions based on priorities, goals, and constraints. Once improvement projects are defined and approved, action plans must be defined to put them in place. The importance of improvements must be communicated to all maintenance engineers. To achieve this objective, the maintenance organization must develop insights into the strengths and weaknesses of the current processes and of its supporting infrastructure.

### 4.2.1 Goals of this KPA

1. Identify improvements to the software maintenance processes by obtaining information from many sources, as well as comparative data.
2. Identify and deploy, in all types of maintenance services, exemplary practices that have proved successful in other organizations.
3. Establish improvement priorities, taking into account the priorities of customers, users, maintenance personnel, and sponsors (IT management).
4. Establish an improvement plan that includes all the maintenance organizational units.
5. Train personnel on improvement concepts and techniques.
6. Ensure that all personnel participate in the improvement efforts and ensure intergroup coordination.
7. Acknowledge contributions to improvement and quality in general.

### 4.2.2 Links with Other KPAs

Refer to the Maintenance Process/Service Definition KPA for more information on the categories of services and maintenance processes and related infrastructure [SEI 2002, PA152.R101].

Refer to the Process, Service, and Software Quality Assurance KPA for more information on the independent reports on process review, services, and software for maintenance that constitute useful information for process improvement [S3<sup>m®</sup>].

Refer to the Causal Analysis and Problem Resolution KPA for information on techniques used in the causal analysis of defaults, failures, and other maintenance problems [S3<sup>m®</sup>].

### 4.2.3 Expected Results of this KPA

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- Senior management provides tangible support, resources, and adequate funding to ensure the viability and efficiency of the activities of software maintenance process improvement.
- The responsibility for process improvement is institutionalized and (a) is assigned first to a coordinator and then to a specialized function, (b) positioned at an independent organizational level, rather than within an existing operational unit, (c) a repository of processes and measures is established and available for decision-making, and (d) while current operational budgets are used for improvement, additional budgets are progressively dedicated to process improvement.
- The software maintenance unit implements a process definition and a process improvement plan aligned with the improvement plans of IS/IT and other governing organizations.
- The maintenance processes of other IS/IT organizations are assessed on a regular basis as a way to determine their strengths and weaknesses.
- The evaluation of the maintenance processes leads to the definition and deployment of action plans that are managed in an increasingly quantitative manner.

## 4.3 MAINTENANCE PROCESS/SERVICE DEFINITION KPA

The Maintenance Process/Service Definition KPA requires the identification, design, and maintenance of standardized processes. There is also a requirement to develop an infrastructure to support these processes; this includes a description of the specialized request life cycle for each type of maintenance (operational support and evolution), the policies and criteria for process customization, the organization's process repository, and the library in which the documentation about processes resides.

The objective of this second KPA is to establish, document, and maintain a current and usable repository of standard processes for software maintenance and the infrastructure linked to these processes (ISO 14764, s. 6.1). A guide for tailoring standard software maintenance processes must be developed. The standardized maintenance processes must be communicated and a repository must be accessible.

### 4.3.1 Goals of this KPA

1. Identify key software maintenance processes, activities, and services.
2. Generalize and standardize software maintenance processes/services.

3. Establish guidelines for standardized maintenance process/service tailoring.
4. Communicate standardized software maintenance processes/services.
5. Establish a repository of standard maintenance software processes/services.
6. Integrate the software maintenance processes/services with those from other IS/IT organizational units (see Figure 3.1), especially where there is a direct interface.

### **4.3.2 Links with Other KPAs**

Refer to the Maintenance Process Focus KPA [SEI 2002, PA153.R101] for more information on process improvement and the support required for process definition.

### **4.3.3 Expected Results of this KPA**

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- The identification, definition, and tailoring of processes/services is achieved based on international standards and guidelines relevant to software maintenance organizations.
- The definition and standardization of key maintenance processes/services such as:
  - SLA
  - Software transition
  - Operational support services
  - Evolution and corrective services (e.g., adaptive, preventive, perfective, and corrective)
  - Problem resolution
  - Involvement in software development or procurement projects
  - Maintainer involvement in application software data recovery activities after a failure
  - Production system surveillance
- The existence of a directory of software maintenance processes/services and procedures. This library is used as a reference for documents pertaining to the maintenance organization.
- The existence of a documented rationale specifying which group is doing maintenance and what services are offered for all application software.

## **4.4 MAINTENANCE TRAINING KPA**

The Maintenance Training KPA first requires the identification and assessment of needs for training new personnel in software maintenance management, in both

technical activities and maintenance service. The goals are to develop competencies and skills of both software maintenance engineers and their managers. This will allow them to efficiently carry out the responsibilities and roles that have been assigned to them [SEI 2002, PA158]. Other maintenance skills are more effectively learned by the mentoring of newcomers by more experienced engineers. In each case, the best means of training are planned, selected, and used.

The objective of this third KPA is to identify training needs and training plans, and to identify the profile of the required personnel for new software in the pre-delivery stage. Individual commitment of the personnel to their own education and training is essential and must be promoted. New employees must be well trained and supervised as they become integrated into the maintenance team. Training must include process training as well as software and technology training. In this KPA, the training of users of the software products and maintenance services is included.

#### **4.4.1 Goal of this KPA**

1. Identify, request, and obtain the resources required for training and education of maintenance engineers.
2. Harmonize corporate training and locally planned and funded maintenance training.
3. Ensure that there are competent and motivated maintenance personnel.
4. Motivate maintenance engineers by promoting education and training on processes, software, and technology.

#### **4.4.2 Links with Other KPAs**

Refer to the Maintenance Process/Service Definition KPA for more information on maintenance process categories and their infrastructure [SEI 2002, PA158.R101].

Refer to the Maintenance Planning KPA for more information on specific needs for training, depending on the various maintenance services [SEI 2002, PA158.R102].

#### **4.4.3 Expected Results from this KPA**

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- Senior management commits and provides enough resources and funds to ensure efficient delivery activities for the training of software maintenance engineers.
- There is a well-balanced training plan that addresses:
  - Maintenance task definitions and career tracks, which are well supported by processes specific to software maintenance

Rewards programs recognizing the performance of maintenance employees  
Maintenance processes, as well as their technical aspects  
Development processes requiring the involvement of maintenance engineers in project support and in software transitioning  
Practical training on the software portfolio and related data structures and business rules  
Ongoing improvement deployment  
Motivation as an important factor in staff productivity and customer satisfaction  
Ongoing training  
An assessment to verify that individual training has been carried out and that annual goals have been reached

## **4.5 MAINTENANCE PROCESS PERFORMANCE KPA**

The measurement of maintenance process performance requires first that the processes having the greatest impact on quality and performance be identified. Once identified, measures must be associated with process performance. It is important to identify and implement appropriate measures and determine a baseline for process performance. Once current performance has been identified, an improvement/performance goal can be set. The process performance measures can also be useful for building prediction/simulation models.

The objective of this fourth KPA is to obtain and maintain a quantitative description of the performance of the key activities of the software maintenance processes. This will help in explaining current process/service performance and setting quality targets/goals. Quality goals are set by communicating and negotiating the proposed quality targets/goals, taking into account the current achievement of service levels [SEI 2002, PA164].

### **4.5.1 Goals of this KPA**

1. Identify the processes and key activities of software maintenance that will be subject to performance analysis.
2. Set up a performance baseline for maintenance processes.
3. Identify and set up measures for software maintenance process performance.
4. Set up models for predicting process performance.

### **4.5.2 Links with Other KPAs**

Refer to the Quantitative Management road map of the Maintenance Process Performance KPA for more information on the use of a performance baseline for a process and its models [SEI 2002, PA164.R101].

Refer to the Maintenance Measurement and Analysis KPA for more information on the specification of measures of software maintenance, data collection, and data analysis [SEI 2002, PA164.R102].

### 4.5.3 Expected Results from this KPA

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- Quality and performance measures are established for products, intermediate products, and key application software.
- Measures are harmonized for all the organizational units for software maintenance and are integrated into the IS/IT measurement program.
- Measures have a goal and a baseline.
- Measures are split into two categories: (1) external measures, which are those seen by customers, (2) internal measures, which are more technical and focus on products and software characteristics.
- Evaluation of processes implies the setting up and deployment of action plans.
- Measures are collected at the operational level and reported in the repository for later analysis of software maintenance.

## 4.6 MAINTENANCE INNOVATION AND DEPLOYMENT KPA

Innovations in maintenance processes and technologies and the deployment of incremental and innovative improvements to them requires first the establishment of selection criteria for the analysis of possible improvement alternatives. Projected returns on investment can be verified through pilot initiatives before improvements are deployed. The introduction of innovations and improvements must be planned, managed, and controlled to ensure that there is objective evidence that the expected benefits can really be achieved.

The objective of this fifth KPA is to identify and deploy innovations and improvements that will have a measurable and significant impact on the performance, quality, and profitability of software maintenance processes/services and technologies (minor improvements are not addressed here). Selection criteria are identified to assess the various competing proposals. Expected benefits are analyzed in comparison with the required investment. These improvement activities support the process performance and quality goals derived from the business goals of the organization [SEI 2002, PA161].

### 4.6.1 Goals of this KPA

1. Identify the maintenance improvements and innovations with the greatest potential.

2. Conduct pilot projects to verify the performance of the most promising alternative.
3. Identify, from the pilot projects, the improvements to be deployed.
4. Plan the improvements, manage their deployment, and measure the benefits.

### **4.6.2 Links with Other KPAs**

Refer to the Maintenance Process/Service Definition KPA for more information on the deployment of improvements at the organizational level [SEI 2002, PA161.R101].

Refer to the Maintenance Process Focus KPA for more information on the source of improvement feedback, data collection, management of process improvement proposals, and co-ordination of activities for improvement of daily activities in software maintenance [SEI 2002, PA161.R102].

Refer to the Maintenance Training KPA for more information on the training that supports the deployment of process or technology improvements [SEI 2002, PA161.R103].

Refer to the Maintenance Process Performance KPA for more information on the quality goals for the execution of processes, the process performance models, and the formal evaluations of improvement and innovation proposals. The quality goals can be used to analyze, prioritize, and choose among the suggested improvement proposals. The performance models can be used to quantify the planned impact and the expected benefits of proposed innovations. Baselines can also be used to analyze the current versus expected performance of a process [SEI 2002, PA161.R104, PA161.R108].

Refer to the Maintenance Measurement and Analysis KPA for more information on the specifications of software maintenance measures and their collection, as well as data analysis and how to communicate the results [SEI 2002, PA161.R105].

Refer to the Event/Request Management KPA for more information on the co-ordination of the deployment of improvements to process and technology in the operational processes of software maintenance.

### **4.6.3 Expected Results from this KPA**

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- Improvement initiatives are identified and deployed [SEI 2002, PA161. N101]:
  - Software quality improvement (e.g., functionality, performance, and maintainability)
  - Productivity improvement
  - Faster execution times
  - Increased customer satisfaction

- An infrastructure is deployed that fosters all maintenance resources to propose improvements and innovations.
- The organization has the capability to evolve and deploy improvements and innovations to software maintenance processes, product, and platform.
- Personnel can participate actively in the improvement and innovation of software maintenance.
- Proposals, from all sources, are systematically collected and analyzed.
- Pilot projects are carried out, prior to a more global deployment, in order to evaluate proposals that include major changes, unproven technologies/techniques, or high-risk activities.

## 4.7 SUMMARY

In this chapter, the first process domain of S<sup>3</sup>M®, including its relationships with other process domains, was introduced. The KPAs were described in greater detail. The goals were stated, as well as the expected results from implementing the processes associated with each KPA. The next chapter describes the second process domain, “Event/Request Management,” which contains four KPAs.

## 4.8 EXERCISES

1. Explain the concept of the process focus on software maintenance and its main goals.
2. The maintenance group accepts and uses audit results. What is the maturity level of this practice?
3. You observe that the improvement initiatives in your maintenance group have their own budget and are managed as specific projects. Explain at which level you are positioned within the S<sup>3</sup>M®. Describe what groups at lower and higher levels do.
4. You have received training on generic processes, but you always have difficulty adapting them to practical situations. At what maturity level are you currently, and which higher-level practices would help you address this problem?
5. You must establish procedures to acknowledge, record, and monitor problem reports and change requests. Which practice can help you do this? Describe what is recommended, and list the relevant references.
6. You provide production accesses to the new employees. Identify at what level of maturity you are currently. Identify what must change in order for you to reach level 3 of this practice.
7. What are the differences between training, experience, and education in terms of their contributions to process maturity?
8. Explain why maintenance work can be more challenging than that of development. Also, explain why maintenance employees require a good interper-

sonal communication skills set. What skills set would a good maintenance employee require?

9. Have a look at a copy of a program (or class) you did not write that has over 1000 lines of code. Evaluate its internal documentation, and compare it with written documentation. It is possible that there is no written documentation! If there is, evaluate its correctness. If you became responsible for this program, what additional documentation would you like to have? What is the impact of (a) the size, (b) the complexity, and (c) the choice of variable names of this program on your ability to maintain it?
10. Your boss publishes maintenance measurement results in terms of:
  - Measures as a percentage of a satisfaction survey
  - Time sheets to record effort by activity/resources
  - Number of service calls of all kinds
  - Number of usersAt what maturity level is this maintenance measurement and what would be the next stage required to improve the situation?
11. You are asked to use estimation models rather than your own experience to prepare project estimates. Which practice(s) can help you do this?

# Chapter 5

---

## Event/Request Management Domain

### This chapter covers:

- An overview of the S3<sup>m®</sup> second process domain: Event/Request Management
- For each of its key process areas, the objectives, goals, expected results, and links with other KPAs:
  - Event/Request Management
  - Maintenance Planning
  - Monitoring and Control
  - SLA and Supplier Agreement Management

This chapter presents an overview, goals, and objectives of the second process domain of the proposed model. The detailed practices for the zero, first, and second maturity levels of this second process domain are presented in Chapter 9.

### 5.1 OVERVIEW

This second process domain covers the management of all types of software maintenance events and requests. An adequate management system for software maintenance must satisfy all the criteria of SLAs, as well as those of the technical domain.

This process domain covers four KPAs:

Req1: Event/Request Management

Req2: Maintenance Planning

Req3: Maintenance Request/Software Monitoring and Control

Req4: SLA and Supplier Agreement Management

Figure 5.1 represents the interactions between the Event/Request Management KPA and the other process domains of the model.

The Event/Request Management KPA is the entry point, through the help desk, for daily communication with the users. All types of user requests (e.g., operational support, MRs, or PRs) are handled through this KPA, which must quickly identify customer priorities and master the organization's problem resolution process. It is through the many interfaces of the problem resolution process that all the external operational communications are coordinated. The goal of this KPA is to ensure that the software maintenance processes/services meet the agreed-upon SLA quality/service objectives.

The Maintenance Planning KPA helps to plan the allocation of resources to individual requests and to handle rapidly changing priorities. The overall view of all the requests and maintenance tasks is found in this KPA, which sorts requests according to the priorities and availabilities of resources. It handles requests from (a) customers and users, (b) development teams, (c) subcontractors, and (d) computer operations. Once prioritized and authorized, the work items are assigned to software maintenance engineers.

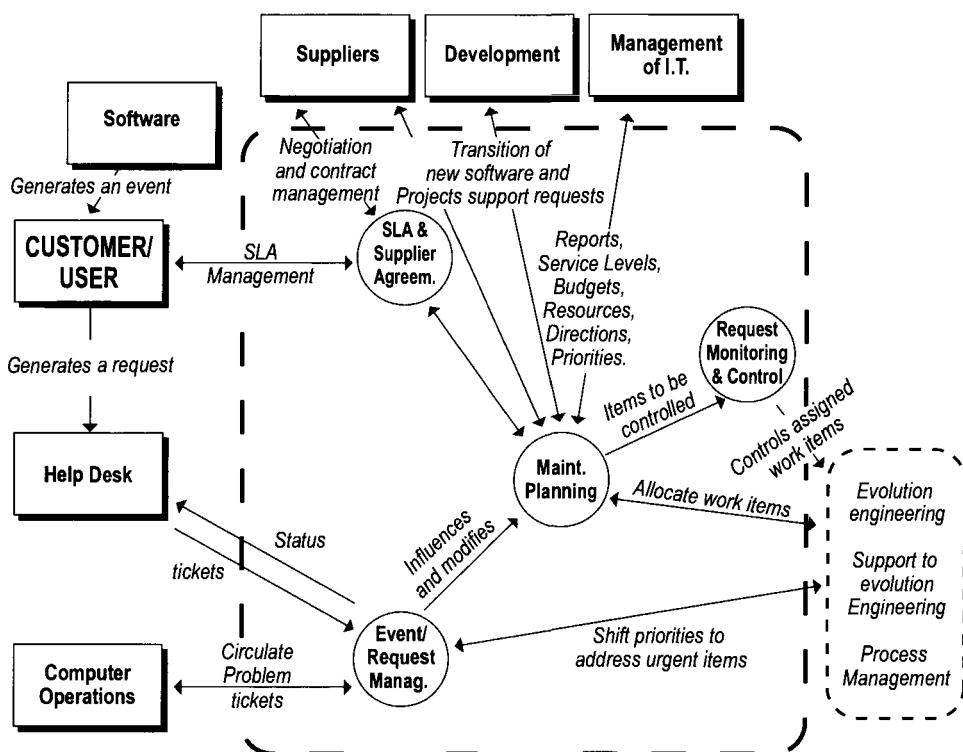


Figure 5.1 Event/request management KPA interactions. Adapted from [SEI 2002].

The Maintenance Request Monitoring and Control KPA identifies items of work that need to be controlled and the individual work items assigned to resources that are currently in progress.

The SLA and Supplier Agreement Management KPA identifies the levels of services and resource requirements, and negotiates prices with customers and service partners. In this KPA, the software maintenance services are described and explained to ensure that the customers understand and are satisfied with the agreements. Software maintenance engineers will also ensure that the suppliers provide an added-value partnership and join in the effort to meet the service-level targets.

## 5.2 EVENT/REQUEST MANAGEMENT KPA

The Event/Request Management KPA ensures that the failures and requests for operational support and software evolution are proactively identified, prioritized, and rerouted, or processed in a way that meets (or exceeds) the terms of the customer's SLA. An event, if it is not resolved, may result in failure to meet an SLA target and, possibly, be the cause of complaints about (a) slow response time in processing a specific request, or (b) not providing an agreed-on service level (e.g., availability or response time).

The objectives of this KPA are (a) to ensure that requests for services and events are identified proactively, (b) to identify the relative importance of a new event versus the ongoing work, and (c) to ensure that maintainers work on agreed-on priorities (to ensure that the terms of the customer's SLA are met). It is also important to communicate proactively about failures, the unavailability of software, and scheduled downtime (e.g., preventive maintenance). Customers must be aware of, and agree with, the status of each event and request. Monitoring of the software in production and its operational infrastructure is key to detecting potential problems early. Maintainers will monitor the behavior of the software (availability, performance, reliability, stability, and status of batch tasks) after a change has been made to ensure that it is stable. Maintainers will stop work on evolution requests when a high-priority event is identified.

### 5.2.1 Goals of this KPA

1. Proactive identification of events and service requests (internal and from customers).
2. Complete and accurate identification, communication, and authorization of priorities of requests (i.e., new events, requests in processing, and requests in service queues).
3. Ensure that maintenance engineers are working on agreed-on priorities.
4. Monitor the behavior of software and its infrastructure (especially following a change).

5. Interrupt current work when a high-priority event occurs.
6. Proactively communicate all events and their planned resolution times.

### **5.2.2 Links with Other KPAs**

Refer to the Impact Analysis road map of the Maintenance Planning KPA to obtain more information on the activity of impact analysis that divides a request into individual work items, enabling the establishment of solution alternatives and the management of the maintenance requirements [SEI 2002, PA163.R101 and R102].

Refer to the Software Evolution and Correction Services KPA to obtain more information on the allocation of specifications to detailed design that will be used for programming activities [SEI 2002, PA163.R104].

Refer to the SLAs and Supplier Agreements KPA to identify the priorities of competing systems, service level targets/goals and procedures for escalation [ITIL 2007b, 4.2].

Refer to the Causal Analysis and Problem Resolution KPA to obtain more information on the process that IS/IT support groups use to deal with an event [ITIL 2007d, 7.2].

### **5.2.3 Expected Results from this KPA**

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- The maintenance work performed is centered on customer/user priorities.
- The interruptions in ongoing work are justified according to the terms of the SLA and agreed on with the customers.
- The maintenance organizational unit meets its levels of service.
- An active monitoring approach, for rapid reaction, is being implemented.
- Information on failure and maintenance activities that affect the customer is quickly circulated.

## **5.3 MAINTENANCE PLANNING KPA**

Maintenance planning ensures the creation and updating of plans that describe the current and planned software maintenance activities [SEI 2002, PA163].

Software maintenance, being a large portion of IT costs, needs to be part of the strategic IT plan [IT Governance Institute, 2007]. Software maintenance planning is typically elaborated from three perspectives: organizational, tactical, and operational. At the organizational level, planning goals are strategic, general, and corporate, and include partners, the SLA, contracts, and licensing. At the tactical level, the planning goal is to identify and plan the activities of predelivery and transition

for new software and also the yearly plans associated with a specific customer account. Finally, at the operational level, the goals are specific: (a) to conduct impact analysis on a given maintenance request, (b) to plan the recovery/failure tests (documented in the disaster recovery plan of individual application software), (c) to plan software versions/releases (e.g., assign each request to a future version/release of the software), and (d) to plan software upgrades (especially the COTS) for the many platforms [Sahin 2001].

All these perspectives of software maintenance planning document the maintenance services demand (that is, the set of requests) for software maintenance [SEI 2002, PA163.N102]. This planning data must be studied to produce a document explaining the allocation of budgets and resources according to the priorities of the demand for services. The planning includes an estimate of the effort needed to process the various requests that translate into needed resources [SEI 2002, PA163.N103]. This maintenance plan must be subject to review and authorization before approval. The software maintenance environment changes quickly. Plans must remain flexible to adapt to the evolution of the situation [SEI 2002, PA163.N101]. Finally, there is a need to acknowledge the close link required between the capacity plan produced by computer operations and maintenance planning [ITIL 2007, 4.3].

### **5.3.1 Goals of this KPA**

1. Ensure a proactive collection and documentation of customer/user, development, and computer operation requests (in the short term, medium term, or long term) to conduct detailed planning.
2. Identify, communicate, and obtain consensus on priorities of requests currently in queue and assigned.
3. Identify the required controls for each type of maintenance service and application software under maintenance.
4. Plan the recovery/failure tests for all software in maintenance.
5. Identify version management plans (sometimes also called release plans) for all software maintained.
6. Prepare software upgrade plans.
7. Establish a rationale for the allocation of new requests to maintenance engineers and for the capacity of maintenance.
8. Inform the stakeholders of maintenance work performed, waiting in queue, and in progress based on agreed priorities.
9. Inform stakeholders on the status of budgets and on the use of resources.

### **5.3.2 Links with Other KPAs**

Refer to the Predelivery and Transition Services KPA to understand planning at the tactical level (and the maintenance plan details) [S3<sup>m®</sup>].

Refer to the Software Evolution and Correction Services KPA to obtain more information on planning across the modification stages [SEI 2002, PA163.R104].

Refer to the Configuration and Version Management KPA to obtain information on the management and control of the changes required to plan for a new version/release [ITIL 2001, 2.8; 2007c, 7.3, 4.4].

Refer to the Event/Request Management KPA to obtain more information on the source of changes and their priorities that have an impact on the planning of a new version [ITIL 2007d, 4.1 and 4.3].

### **5.3.3 Expected Results from the KPA**

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- A sound forecast is available on needed resources in software maintenance (strategic plan, tactical plan, and operational plan).
- SLA, agreement, license, and contract renewals are planned.
- The failure/recovery tests are planned for each software in maintenance.
- Requests are allocated to future versions of the software.
- Software upgrades are planned.
- Predelivery and transition services are planned.
- There is an optimal allocation of maintenance resources to work items.
- Software maintenance has a plan, and this plan is communicated and approved.
- The stakeholders are informed about the plans.

## **5.4 REQUEST/SOFTWARE MONITORING AND CONTROL KPA**

The maintenance request (and services) monitoring and control KPA includes surveillance of work progress versus estimates, commitments, and plans. It also includes surveillance of the service levels of software in operation. When there is a deviation between the actual service level and the planned service level, action must be taken to correct the situation [SEI 2002, PA162].

The objective of this KPA is to ensure that the service delivery targets and also the application software service levels are respected. Maintainers must proactively monitor the gaps in service levels, make needed adjustments, and review/establish new agreements for delivery or service levels when needed.

### **5.4.1 Goals of this KPA**

1. Control the progress of the various maintenance work items and their software service levels.

2. Identify gaps between actual performance and agreed-on service levels.
3. Agree with the customer/user to review and amend service delivery and service levels based on events and performance.

### **5.4.2 Links with Other KPAs**

Refer to the Maintenance Planning KPA to obtain information on maintenance planning [SEI 2002, PA162.R101].

Refer to the Verification and Validation KPA to obtain more information on the technical reviews and management reviews for controlling progress and the milestones of maintenance work items [S3<sup>m</sup>R].

Refer to the Maintenance Measurement and Analysis KPA to obtain more information on the process of measurement and analysis and how to report the information [SEI 2002, PA162.R102].

### **5.4.3 Expected Results from this KPA**

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- Maintenance activities are performed according to forecasts from the various levels of maintenance planning.
- Maintenance work items, costs, and respect for commitments are tracked.
- Activities are reviewed and replanned when they progress differently from what was originally planned.
- Responsibility for communication of rectifications, commitments, and promises is known by the resources.
- Reviews are conducted with the customers at main checkpoints to examine the results and plans, and also the progress of work.
- Corrective actions are taken if it is determined that there is a difference between results and the forecasting accepted by customers.

## **5.5 SLA AND SUPPLIER AGREEMENT MANAGEMENT KPA**

This concept of SLA and Supplier Agreement Management applies when maintenance is not handled as a project. The SLA and Supplier Agreement Management KPA requires that maintenance engineers master the many specialized contractual agreements found in software maintenance. There is a need to define the SLAs and various other contracts to proactively manage the interfaces with customers and suppliers [SEI 2002, PA166].

There are two main objectives for this KPA. The first concerns customer account management and is met by establishing and tracking the service levels formally.

Agreements (reached in the previous KPA) describe the terms of services and make official (a) the number of maintenance services that have to be delivered, (b) the cost of each service, (c) the quality and efficiency of the products and services, and (d) the establishment of standardized measures for performance measurement and reporting to customers. The second objective is for the establishment of individual contracts with suppliers to define the reciprocal commitments linked to the SLAs [ITIL 2007b, 4.2]. Procedures for execution and tracking of SLA and contract performance with suppliers are also needed.

### **5.5.1 Goals of this KPA**

1. Agree on what maintenance services are required for both software and customers.
2. Establish the SLAs with each customer and obtain commitments from suppliers (i.e., sometimes outsourcers) for the software maintenance services.
3. Implement the processes and assign the roles/resources to ensure the respect for agreements.
4. Assess periodically the service performance and discuss the results with the customer.
5. Implement a documented procedure and policies to assess the performance of the maintenance services and its suppliers.

### **5.5.2 Links with Other KPAs**

Refer to the Requests/Software Monitoring and Control KPA to obtain more information on the tracking of key activities for maintenance [SEI 2002, PA166.R101].

Refer to the Predelivery and Transition Services KPA to obtain more information on maintenance plans that analyze, identify, and describe the services required for new software. This KPA also determines the organization that will be in charge of maintenance services [SEI 2002, PA166.R104].

### **5.5.3 Expected Results from this KPA**

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- The agreements are defined, formalized, and negotiated.
- The agreements and responsibilities are defined.
- The SLAs are expressed in words that are easily understood by the customer.
- A communication channel exists, is used, and works to ensure agreements are performing well.

- The agreements and contracts evolve with situations.
- An objective assessment of services and suppliers is performed.

## 5.6 SUMMARY

In this chapter, the second process domain of S3<sup>m®</sup>, including its KPAs and the relationships between them and with other process domains, was introduced. Each KPA was described in greater detail, identifying its goals as well as the expected results from implementing its processes.

The next chapter describes the third process domain, the Evolution Engineering Domain, which contains four KPAs.

## 5.7 EXERCISES

1. Customers phone you to complain that their system has been taken out of service without their approval. Moreover, when the system is not available to them, they never know whether this is due to a failure or is a regular maintenance problem. At what level of maturity is your maintenance group currently? Which practice, if implemented, will help address this situation?
2. You notice that, since the outsourcing, there are an increasing number of modification requests. Your new boss explains that you must be more flexible and work harder. The developers have prepared a list of small projects (to take 30 days or less, according to them) and want to you to do these jobs. Explain which process, unique to maintenance, is at risk here, and what consequences can be expected if you allow this situation to remain out of control.
3. Jimmy joins you for lunch and talks to you about a problem ticket that is being shuffled back and forth between him and the data processing department. Jimmy is positive that his investigation is complete, and that it is not a software problem but rather something in the infrastructure of the Web servers that is causing the problem. You check the system monitoring the problem requests and you find that the ticket no longer exists. Which practice has been bypassed? What are the consequences of this action? At what level of maturity is this process?
4. After having talked with Mario of the data processing department, and agreed with him that the disappearance of a problem ticket is not acceptable, the problem ticket reappears the next morning. Jimmy prints a copy of it, which you examine. The problem ticket provides the following information:
  - The request has a single number.
  - The request has a low priority.
  - The request does not have a category.

- The name (or the acronym) of the software and the version for which a software problem was detected are not registered.
- It is impossible to tell in which environment the software is being used.
- The date and time the problem occurred, and when the modification request was received.
- The person who submitted the request/problem is not identified.
- The title of the request is “Bug in transaction L3.”
- The effects and consequences of the problem are described as follows: “When the date is put into the L3 transaction, the screen goes black.”
- The causes of the problem are not documented.
- The fields “planned effort” and “actual effort” are empty.

Specify what information is useful on this problem ticket. Specify what information is missing from this problem ticket. Identify the practice that would be useful for improving this situation.

5. The customer who issued this problem ticket is becoming restless. Unbeknownst to you, it is the director of finance who uses this transaction! He phoned your boss, who wants to know how long the L3 transaction had not been working before the problem with it was reported. Which practice should have been put in place to address this issue? At what level of maturity would you have been if you had been able to answer this kind of question quickly?
6. The director of finance was not very pleased when he learned that his request had the lowest possible priority. It is the end of month and the financial figures must be produced for the board of directors. What practice was not observed here? At what level of maturity is the activity of assigning priorities and communicating with the customer?
7. You return to your office and let Frank deal with this problem since he has a meeting this afternoon to discuss the transition of a new project from development. You know that the development team will try to hand over the project on June 1 (that is, in six months). Explain the format of the transition planning document, as well as proposals you can make to minimize (a) the number of maintenance programmers, and (b) the number of maintenance problems.
8. After having helped Frank get out of trouble, you proceed with writing the e-mail that you promised to send to the project leader about planning the first project management review as a follow-up to the project feasibility analysis that was recently tabled. Describe the content of this e-mail, as well as the distribution list. What practice can help you with this? Which standards support your request?
9. Yesterday, you copied your boss on your e-mail, and your boss is glad to see that you will be conducting a project review prior to the final transition. You now have to explain to him that there will be three others! He asks you

to explain what will occur in these other follow-ups and control reviews. Describe briefly what will occur before the system is transferred to you. What practice will help to you in this? Which standard supports your approach?

10. You have difficulty understanding the management style of your new employer. You decide to look into the process of setting up service agreements. Your boss explains to you that over the years this has required a lot of negotiations with the city and with government officers. You ask for details, and he explains his approach this way: "The contract is simple in and of itself: we make corrections and modifications, and they pay on a monthly basis. When there is a problem, I deal with it personally. We have set up an outsourcing outfit in a third country to do the work at lower cost, and in addition we can benefit from time zone differences. Finally, the work output is reviewed by us. It is as simple as that!" What is the maturity level of this service agreement? What practices are involved here? Since you have been tasked to develop a new service agreement (and you are almost finished doing it), justify the approach you adopted to propose improvements to current practices.
11. The next day, you drop in to see Frank for an update. Frank is in a great mood. He has followed your advice: the correction is in place and the customer is satisfied! The director of finance asked him why the request went back and forth so many times between the data processing and maintenance departments without anyone doing anything about it. Clearly, a more effective problem resolution process is required. Didier is working on a solution. What practice would be of help to him? Which support document should he be looking at for further information on this topic?
12. Your boss introduces you to the consultant who will now be in charge of the software maintenance account. You ask him what approach he wishes to use with the customers, and the answer is: the centralized approach. What do you think about it? What practice can help you in discussing this topic?
13. The team leader in charge of support for the financial software system informs you that your ERP representative always needs two days to solve a problem, while your customer service agreement specifies a maximum turnaround time of three hours! By the time your ERP rep comes back with a solution, the problem has always escalated. Your manager wants to know why the service level agreement targets for the ERP system are never met. What is happening here, and how can this problem be solved permanently? What is the maturity level of this practice? If you cannot change the contractual agreement, which practice can still help?
14. You ask the team leader in charge of support for the financial software system to show you the details of the description of the ERP software that appears in the service agreement. He tells you that there isn't one! You write one with him (a page and a half in length). What practices will help you? Which reference document can help you? Prepare to present your document to the class.

15. Your new company is looking for a way to invoice maintenance services. Currently, charges are based on the number of hours worked, plus a profit margin. Describe candidate invoicing strategies, and indicate which documents can help set out the relevant details.

# Chapter 6

---

## Evolution Engineering Domain

### This chapter covers:

- An overview of the S3<sup>m®</sup> third process domain: Evolution Engineering
- For each of its key process areas, the objectives, goals, expected results, and links with other KPAs:
  - Predelivery and Transition services
  - Operational support services
  - Evolution and corrections services
  - Verification and validation

This chapter presents an overview and goals of the third process domain of the proposed maturity model. The detailed practices for the zero, first, and second maturity levels of this third process domain are presented in Chapter 10.

### 6.1 OVERVIEW

This third process domain covers operational activities pertaining to software maintenance. It contains practices associated with predelivery, transition, operational support, evolution and correction services, and verification and validation.

This process domain covers the following four KPAs:

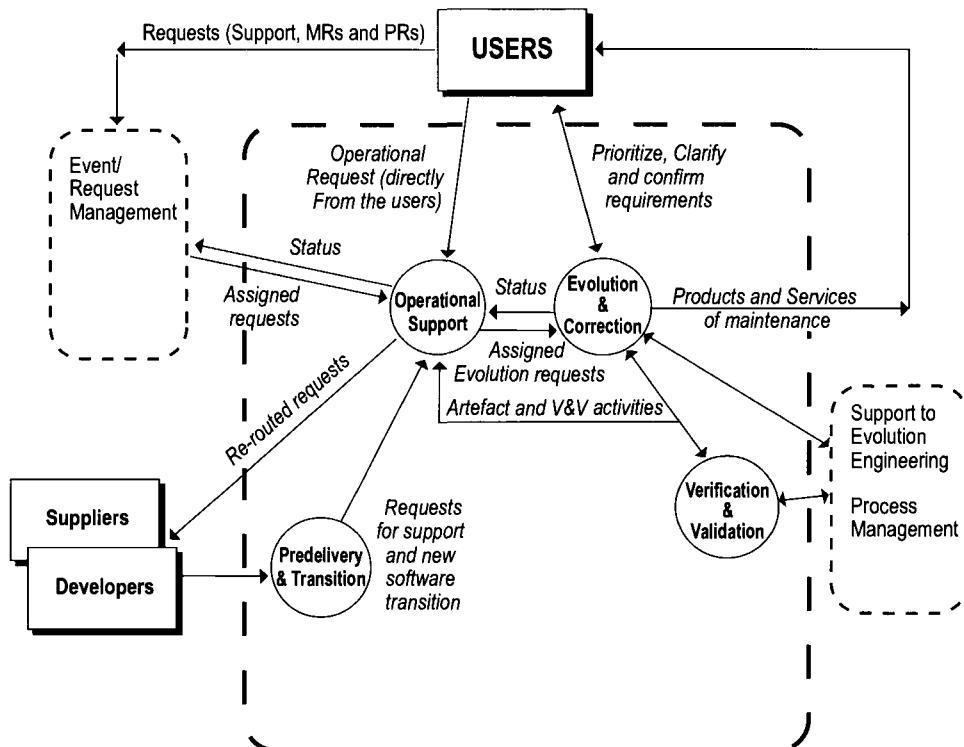
- Evo1: Predelivery and Transition Services
- Evo2: Operational Support Services
- Evo3: Software Evolution and Correction Services
- Evo4: Verification and Validation

Figure 6.1 shows the interactions between the four software evolution engineering KPAs and the other process domains.

The Predelivery and Transition Services KPA should be the first contact of the maintenance engineers with the software they will have to maintain in the future. The main objective of this KPA is to increase the maintainer's understanding and try to influence the maintainability of the software during its development (if possible).

The Operational Support Services KPA handles service requests that were not solved by the "super-user" or the help desk. A super-user is the name given to a user who has been assigned the task of supporting and training other users of the software. Operational support services does not modify the software but provides a consultancy service for it:

- Production of a one-time extract of information from existing systems
- Provision of technical information and counseling to developers and operations
- Provision of help to customers for better understanding the software, a transaction, its data, or its documentation.



**Figure 6.1** Evolution engineering KPA interactions. Adapted from [SEI 2002].

The Software Evolution and Correction Services KPA proceeds with a modification to the software. The maintenance programmer will look further into the impact analysis and will produce a detailed design, perform modifications and tests, and provide documentation. In this KPA, specific maintenance processes (and their workflow) are implemented for each type of maintenance service. The evolution processes are similar to the developer's processes (they refer directly to ISO 12207 section 5.3), but are adapted for the maintainer's use.

The Verification and Validation KPA controls the request modifications, which have been assigned by the resources and are currently in evolution.

## 6.2 PREDELIVERY AND TRANSITION SERVICES KPA

This is one of the key processes of software maintenance. It includes a sequence of coordinated activities and formal revisions from which the responsibility for newly developed software (or the acquisition/adaptation/parameterization of COTS or ERP) passes from the developer (supplier) to the maintainer function, which will take responsibility in the form of operational support, as well as for evolution and daily modifications [ISO 2006, 6.9; ITIL 2007c].

The objective of this KPA is to increase the maintainer's understanding and try to influence the maintainability of the software during its development. It includes the following activities:

- Identify, receive, and prepare the transfer of knowledge and training pertinent to the software.
- Try to influence the software maintainability characteristics during its design phase.
- Identify, influence, complete, and collect process and product documentation that will be useful during maintenance.
- Participate in configuration management and testing of the software during predelivery.
- Understand and document the status of each PR during acceptance and transition.
- Establish the details of the SLA for the new software.
- Prepare software environment/maintenance support.
- Carry out the formal phases of the process transition [Pigoski 1997, chapter 11].

### 6.2.1 Goals of this KPA

1. Software maintenance cost reduction.
2. Increase the maintenance resource skills (in terms of quality of service, knowledge, satisfaction, etc.) to deal with the new software.

3. Convince stakeholders to consider the characteristics of software maintainability throughout the design phase.
4. Ensure complete documentation from the point of view of the maintainer.
5. Obtain information on training and the transfer of knowledge.
6. Obtain a clear SLA for the new software, as well as an expected level of customer service. This includes the status and priority of all the PRs transferred to the maintainer.
7. Ensure a complete and stable support/maintenance environment for the new software, which must be initiated early.
8. Ease the process of software transition.

### **6.2.2 Links with Other KPAs**

Refer to the Maintenance Process Focus KPA to obtain more information on improving high-level predelivery and transition activities [S3<sup>m®</sup>].

Refer to the Maintenance Process/Service Definition KPA to better understand the need to define, document and standardize the predelivery and transition process [S3<sup>m®</sup>].

Refer to the Maintenance Training KPA to learn more about the training activities related to software transition. It describes the necessary training on standardized processes and the required adaptation of these processes [S3<sup>m®</sup>].

Refer to the Maintenance Planning KPA to obtain more information on software transition planning [S3<sup>m®</sup>].

### **6.2.3 Expected Results of this KPA**

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- Maintainers influence the software maintainability characteristics during the developers' design phase.
- An SLA exists for the new software.
- A software support/maintenance environment exists for the new software.
- Historical evidence of transition and a record of transition measures exist for the new software.

It is also important to:

- Identify, influence, complete, and collect the documentation needed by the maintainer.
- Communicate to stakeholders the status of all PRs during the final phases of the transition.

## 6.3 OPERATIONAL SUPPORT SERVICES KPA

Operational support comprises all the activities that do not involve a software modification. These activities are, for example, (a) answer questions, (b) provide information and tips, (c) help customers improve their understanding of a transaction, documentation, or a business rule, (d) supervise the software in operation, and (e) create occasional reports (ad hoc). It also includes consultancy services for developers and suppliers who are in need of technical information or data extracts from an existing system.

The objective of this KPA is to ensure a clear definition, identification, and time recording/invoicing of operational support activities. It also needs to highlight adequate maintenance support (after regular office hours) to clarify/register the effort devoted to the various operational support activities. These activities are highly time-consuming and must be recognized, but they are often neglected. The importance of these activities must be communicated to all the stakeholders.

### 6.3.1 Goals of this KPA

1. Clarify and identify operational support service and activities.
2. Enhance the visibility of these services and the added value of operational support activities.
3. Invoice operational support activities.
4. Establish a justification for the allowance of new requests.
5. Communicate the operational support tasks to stakeholders.

### 6.3.2 Links with Other KPAs

Refer to the Event/Request Management KPA to learn about the source of operational support requests and the supervision of application software [ITIL 2007d, 4].

Refer to the Causal Analysis and Problem Resolution and Requests/Software Monitoring and Control KPAs to learn about the source of production problems and more about how to identify the important times and events that require closer monitoring of application software [ITIL 2007d, 4].

Refer to the SLAs and Supplier Agreement KPA to learn about reliability objectives and customer/ software service description [ITIL 2007d, 4].

### 6.3.3 Expected Results of this KPA

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- Support activities are recognized as value-added activities both externally to support end-users/customers and internally to support IT initiatives.

- Software Maintenance is organizing its evolution activities around it.
- The divide between operations, development and maintenance activities has been softened by clarifying maintainers support scope.
- Value-added support activities are promoted as knowledge-transfer activities.

## 6.4 EVOLUTION AND CORRECTION SERVICES KPA

This KPA contains activities/tasks for modifying application software. It consists of detailed design, evolution (adaptive, perfective, and preventive), corrections, and testing activities. The activities use standardized and published maintenance processes, as well as supported methods, tools and environments. The evolution and correction of software may call upon activities that are similar to the activities of the developers [ISO 12207, 5.5.3].

Impact analysis from the maintenance planning KPA serves as an input to this KPA. Starting from the option chosen in the impact analysis, one proceeds with the description of detailed specifications of the required modification. The maintainer ensures the management of customer requirements during the modification. Once the detailed design is obtained and verified, the maintainer changes the software, conducts the unit tests, and performs the documentation changes. For correction services, the maintenance engineers may elect to implement a temporary solution to restore service and then proceed with an MR to solve the issue permanently.

### 6.4.1 Goals of this KPA

1. Software evolution and correction activities must not introduce a defect/failure.
2. The maintainer uses the impact analysis to accelerate the detailed design activities.
3. The maintainer has the tools, the environment, and the mechanisms to support software correction and modification processes in order to improve productivity.
4. The software is modified in accordance with an established procedure to make sure not to degrade it over the years.
5. The tests are carried out in accordance with the established procedure in order to make sure not to degrade the software or to introduce defects and failures.
6. The results of specifications, of detailed design, of implementation, and of testing are mutually coherent and in agreement with the requirements of each request.

### 6.4.2 Links to Other KPAs

Refer to the Maintenance Planning KPA to learn more about the impact analysis activity, which details the solution alternatives and the chosen alternative, as well as the customer authorization to proceed with the change [SEI 2002, PA163.R101 and R102].

Refer to the Maintenance Planning KPA to learn more about the estimates associated with a software maintenance service request [SEI 2002, PA163.R102].

Refer to the Software Rejuvenation Process for improving the software each time we conduct an evolution.

### 6.4.3 Expected Results of this KPA

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- The authorized customer option is implemented.
- The requirements are documented, and any modifications negotiated with the customers are also documented.
- The modification is documented. There is traceability between the changes and the components of the detailed design, as well as between the coding elements and the test records.
- The unit and regression tests are carried out, documented, checked, and revised.
- The maintainer is ready to communicate with the customers to obtain acceptance of the change and plan its release into production.
- Software is not degraded but improved slightly each time.

## 6.5 VERIFICATION AND VALIDATION KPA

The verification and validation activities of the maintainers are similar to those of the developers, but address different issues. Validation demonstrates that a service or a maintenance request, as provided (or as it will be provided), will perform as intended, whereas verification addresses whether or not the work was properly carried out to meet the requirements of the previous stage; for example, requirements, detailed design, and programming. In other words, validation ensures that “the right job was done,” whereas verification makes sure that “the job has been done right.” Also, maintainers specialize in regression testing [ITIL 2007c, Table 4.12]. This is a key “V&V” skill that is mastered in this KPA.

The objective of this KPA is to make sure that the V&V activities are planned, that the established procedure is followed, and that it is demonstrated that the products/services meet the internal requirements and those of the customers. The organization must communicate the importance of the activities related to quality to all its

personnel. The maintenance organization strives to ensure that the defects do not reach the customers.

### 6.5.1 Goals of this KPA

1. Plan V&V activities in order to be certain which ones will be performed.
2. Follow the V&V execution procedure.
3. Ensure that the work maintenance product meets the specified requirements [SEI 2002, PA150; ITIL 2007c, Table 4.12].
4. Show that the products and maintenance services perform as projected when they are used (or moved into operation) [SEI 2002, PA149].
5. Stipulate that both products and maintenance be subject to revision before their delivery to the customers.
6. Discover defects before delivery to the customers.

### 6.5.2 Links with Other KPAs

Refer to the Maintenance Requests/Software Monitoring and Control KPA to understand where V&V is used to ensure that stakeholders (customers, subcontractors, developers) and technical support live up to their promises and meet the process requirements of the organization. Refer to the Follow-up and Maintenance Supervision KPA [S3<sup>m®</sup>].

Refer to the SLAs and Supplier Agreements KPA to learn more about the reviews needed with customers and subcontractors [S3<sup>m®</sup>].

Refer to the Predelivery and Transition Services KPA to learn more about the reviews that take place with the developer during the predelivery, transition, and acceptance test activities of new software that will become the responsibility of technical support personnel [S3<sup>m®</sup>].

Refer to the Maintenance Planning KPA to learn more about the detailed planning of the V&V activities associated with a specific request [S3<sup>m®</sup>].

Refer to the Software Evolution and Correction Services KPA to learn more about the V&V activities performed during detailed design, programming, and unit and regression testing [SEI 2002, PA163.R104].

### 6.5.3 Expected Results of this KPA

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- Techniques of V&V are adapted for maintenance.
- There are procedures describing how to perform V&V in a maintenance context and by service type.
- Adequate regression testing is carried out for every MR.

## 6.6 SUMMARY

In this chapter, the third process domain of S3<sup>m®</sup>, including its relationship with other process domains, was introduced. Each KPA was described in great detail. The goals were stated, as well as the expected results of implementing the processes associated with each KPA.

The next chapter presents the fourth process domain, Support to the Evolution Engineering Domain, which contains five KPAs.

## 6.7 EXERCISES

1. Last year's development projects are now ready to go into operation. However, unlike this year, last year there were no transition processes. There are all kinds of problems with last year's software, and your colleague Henry is siding with the customer against the developers. He is also lobbying the staff to stall on putting the transfer into production. What is your reaction? What practice is at issue here? At what level of maturity does Henry currently work?
2. You feel fortunate that your projects are following the new transition processes when you see what is going on with Henry's projects! It is now time for you to draw up a checklist for a project that is coming up soon for final transition. Describe the items on this checklist. At what level of maturity is this practice?
3. Norman comes to see you to gain a better understanding of what is necessary to move to maturity level two for the software transition. Explain to him the following items: training users and maintenance employees, documentation, configuration management, and problem reports.
4. What are the major changes of philosophy in the transition from level 2 to level 3?
5. You take time to look at the memory size, the telecommunications links, and the disk space of your software in operation. Norman tells you that it is the job of the operations department to check all that. What do you think? What practice is at issue here?
6. You have coffee with Jim, and you discuss the list of change requests that are outstanding for your software. He confides to you that, since you advised him not to perform detailed analyses, his list is getting shorter by the day! What is it that Jim did not understand when you talked to him? What practice does he have to follow for the analysis of change requests at maturity level 2? (You have to have answered question 8 of the previous chapter to understand what is at issue here!). What practice can you use to illustrate the special (but limited) permission given to him for urgent corrections?
7. Your boss summons you to a meeting. You realize that all the maintenance employees have been invited to a presentation to discuss the replacement of the current software maintenance methodology. The new Xtreme mainte-

nance methodology is presented as six times faster, and the time has come to adapt to the new, much more productive WEB and Java technologies. The outside expert making the presentation explains to you that no more programming standards are required, that you can use the OO programming language of your choice, that no more testing needs to be done because it is the customer who will test, and that documentation is history! Read Poole 2001 and Paulk 2002. Use these two references to explain what is wrong with the external expert's interpretation of the Xtreme approach as applied to software maintenance. With which level 2 practices of software evolution/correction is he at odds?

8. Describe briefly what must be done to reach level 3 for the evolution and design of software. Do you agree?
9. List the differences between the S3<sup>m@t</sup> level 2 V&V practices and the ones proposed by Ilene Burnstein's maturity model [Burnstein et al. 1996b]. Highlight which of her practices are relevant to software maintenance?

# Chapter 7

---

## Support for the Evolution Engineering Domain

### This chapter covers:

- An overview of the S3<sup>m®</sup> fourth process domain: Support to Evolution Engineering
- For each of its key process areas, the objectives, goals, expected results, and links with other KPAs:
  - Configuration and version management
  - Process, service, and software quality assurance
  - Maintenance measurement and analysis
  - Causal analysis and problem resolution
  - Software rejuvenation, migration, and retirement.

This chapter presents the overview and goals of the fourth process domain of the proposed maturity model. The detailed practices for the zero, first, and second maturity levels are presented in Chapter 11.

### 7.1 OVERVIEW

The model's last process domain concerns processes that support software maintenance evolution engineering processes. As defined in ISO 12207, support processes are typically created and made available to support operational processes. They are used as required. Some are jointly managed by other organizational units (e.g., developers, computer operations, finance, and HR) because they are often required by all other organizations in the company. This process domain has five KPAs:

Sup1: Configuration and Version Management

Sup2: Process, Service, and Software Quality Assurance

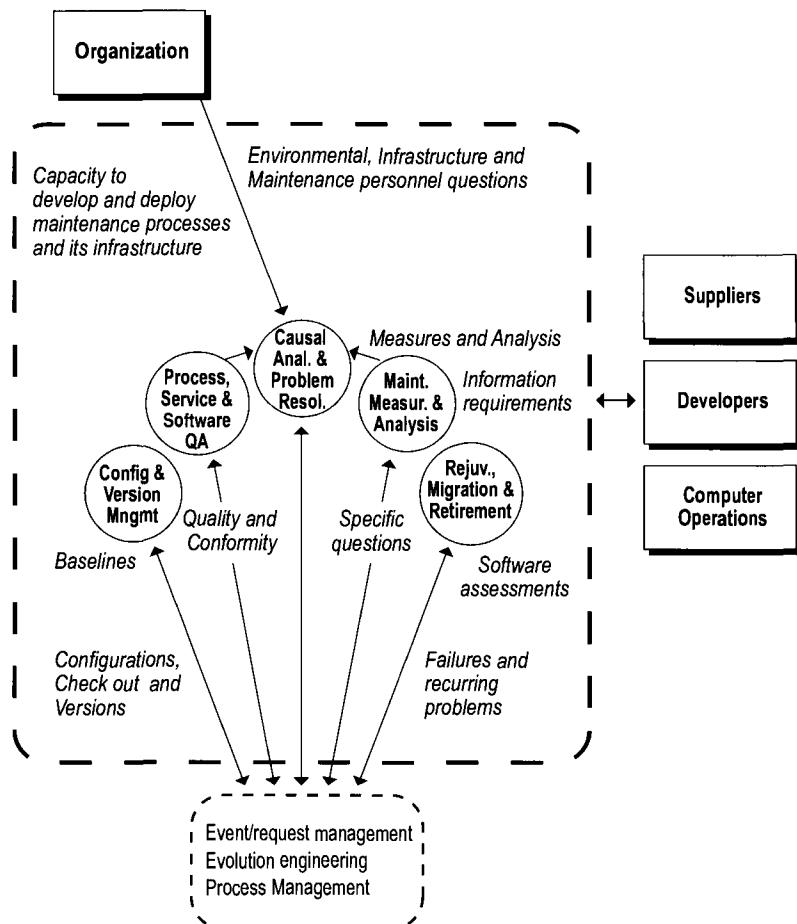
Sup3: Maintenance Measurement and Analysis

Sup4: Causal Analysis and Problem Resolution

Sup5: Software Rejuvenation, Migration, and Retirement

Figure 7.1 represents the interactions between evolution engineering support KPAs and the model's other process domains.

The Configuration and Version Management KPA is a support process that is often shared between the developer and the maintainer within an IS/IT organization. Its primary aim is to establish and control the link between the different intermediate products during a change. It is also responsible for executing the version/release plans using the strategy identified in the maintenance plan [Stark 1997]. Moreover, the maintainer often has the opportunity with configuration management of getting



**Figure 7.1** Support to evolution engineering KPA interactions. Adapted from [SEI 2002].

involved in the predelivery of new software by performing configuration management activities with the developer, in order to:

- Identify, receive, and prepare training and knowledge transfer
- Influence the software's maintainability characteristics while it is being designed
- Identify, influence, complete, and gather useful documentation for maintenance
- Establish and publish the status of the PRs for the various stakeholders
- Establish the details of the SLA for this new software
- Prepare the software maintenance/support environment
- Execute the final transition process' formal steps [Pigoski 1997, chapter 11] or the version/release plan activities if the software is already the responsibility of the maintainer.

The Maintenance Measurement and Analysis KPA groups together all the measurement and analysis activities.

The Process, Service, and Software Quality Assurance KPA groups together all the quality assurance activities of inspections, reviews, audits, and product quality assessments.

The Causal Analysis and Problem Resolution KPA receives information from different processes. This information is required for the study of general questions/situations that need to be taken into consideration in order to solve repetitive problems and guide major improvements and innovations. Problem resolution is a process that involves all the IS/IT support organizations to ensure that good communication exists to quickly find the root cause of a problem [ITIL 2007d, Appendix C].

Finally, the Software Rejuvenation, eMigration, and Retirement KPA optimizes the customer's software to improve maintainability and proposes activities such as:

- Redocumentation
- Software restructuring
- Reverse engineering
- Reengineering
- Software migration
- Software retirement

All these activities use program comprehension techniques to some degree. There are two modes in which these activities can occur. Small-scale optimization activities can be performed when a modification occurs. At that time, there is an opportunity to improve the software because it is opened for a modification. The optimization activity should not have a major impact on the modification effort, and so its scope is often limited.

A second mode of operation, which can have much bigger scope and greater impact, consists of the development of an optimization proposition that is documented and submitted to the customer for their assessment of the potential benefits. Large optimization activities tend to require a project team.

## 7.2 CONFIGURATION AND VERSION MANAGEMENT KPA

Configuration management consists of determining the configuration and description of products and intermediate work products at a precise time. Its objective is to ensure systematic control over configuration changes and to maintain integrity and traceability throughout the life cycle of a request that will modify the existing software.

Configuration management's first objective is to establish a link between the different intermediate products throughout a change. The maintainer is thus able to follow a change's progress from initial request, to requirements, to impact analysis, to detailed design, to programming, to testing, to technical documentation, and, finally, to use by the user himself. This first aspect of configuration is called traceability. Another aspect of configuration management is the use of support software to reserve components while the software is being worked on. This activity ensures that employees cannot change a component without being aware of changes made by a colleague. Developers and maintainers also make images of software at every critical step using the same tools. For example, they would make a copy of the complete configuration at each major step in the testing phase. This technique allows them to isolate an image of all software components at any given time. They can always revert to this state should the work being done become corrupted. It can be imagined that backup copies are, in fact, images of the software's configuration.

The fundamental differences between a backup copy and a configuration image are the following:

- A configuration image contains a great deal of information on individual components.
- A configuration image is kept on-line.
- All employees have access to configuration images and can extract reports from them.
- The configuration image is used during a change.
- A backup copy focuses only on the source code.

Finally, a system's configuration can include its source code, its documentation and the list of all electronic components and base system components that are delivered or in use. This process, and the software that supports it, allows software to be kept up to date year after year.

### 7.2.1 Goals of this KPA

For the maintainer, the goals of configuration management involve the establishment and maintenance of integrity for products and maintenance work products through identifying components and controlling them, maintaining different configuration statuses, and conducting configuration revisions and audits [SEI 2002, PA159]:

1. Identify the components that are part of the software's configuration.
2. Establish and maintain coherence between components during a change throughout the maintenance (configuration) life cycle; that is, maintain its integrity and traceability.
3. Ensure control of software versions and changes in order to retrace them if necessary. Maintain the products, tools, software and procedures required to regenerate application software and its data.
4. Prevent a simultaneous change to the same component by two employees, and thus prevent conflict and loss of productivity. This implies having a procedure and tools to check in and check out items from a configuration management repository.
5. Create a controlled and isolated environment to work in that does not affect developers or customers.
6. Have information on base product content and on the configuration management repository on hand for maintenance purposes.
7. Establish a completely equipped, stable maintenance environment within the first few days of maintaining new software.
8. Create the opportunity to become involved in the transition in order to gather knowledge early in the software life cycle.

### 7.2.2 Links with Other KPAs

Refer to the Maintenance Process/Service Definition KPA to learn more about the need to define the configuration management process [S3<sup>m®</sup>].

Refer to the Maintenance Training KPA to learn more about training activities concerning the configuration management process, and the maintainer's tools and infrastructure in this area [S3<sup>m®</sup>].

Refer to the Maintenance Planning KPA to learn more about plans and environment and configuration management tools for developers. The scope of configuration management services is defined by the developer and maintainer in this KPA as well [SEI 2002, PA159.R101].

Refer to the Maintenance Requests/Software Monitoring and Control KPA to learn more about maintenance request controls using configuration management and infrastructure activities during and at the end of the transition [SEI 2002, PA159.R103].

Refer to the Predelivery and Transition Services KPA to learn more about the maintainer's opportunity to become involved in the configuration management of application software before it is moved on to production [S3<sup>m®</sup>].

Refer to the Software Evolution and Correction Services KPA to learn more about configuration management for a specific change request [S3<sup>m®</sup>].

### **7.2.3 Expected Results of this KPA**

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- The software's maintainability characteristics can be influenced during its design phase.
- The documentation required by the maintainer is identified, influenced, completed, and collected.
- The parties needing to know the status of a PR are informed during the transition's final steps.
- An SLA is developed for each software.
- There exists a software support/maintenance environment.
- Historical reports and records of software transition measures are kept.

## **7.3 PROCESS, SERVICE, AND SOFTWARE QUALITY ASSURANCE KPA**

The Process, Service, and Software Quality Assurance KPA provides the necessary personnel and processes to perform independent and objective reviews and audits of maintenance products and services, as per the organization's established standards and rules [SEI 2002, PA145]. Conclusions from the audits and reviews are given to managers and other responsible parties.

The objective of this KPA is to provide support for delivery of high-quality products and services by giving maintenance engineers a say in the application of the processes, procedures, and rules used in daily maintenance work [SEI 2002, PA145.N102]. This KPA's detailed practices ensure that the standardized processes are planned and used throughout the handling of a request.

### **7.3.1 Goals of this KPA**

1. Ensure maintenance process/service uniformity and application.
2. Objectively evaluate process execution, work products, and services according to the service agreements, standards, and procedures used to meet quality objectives.
3. Identify and document nonconformities.

4. Inform stakeholders about nonconformities and identify ways to eliminate them.
5. Follow up on nonconformity elimination.

### 7.3.2 Links with Other KPAs

Refer to the Maintenance Planning KPA to learn more about the different plans that identify activities, and deliverables that will be subject to quality assurance (e.g., Req2.3.12, 18) [SEI 2002, PA145.R101].

Refer to the Requests/Software Monitoring and Control KPA to learn more about quality assurance controls and milestones [S3<sup>m®</sup>].

Refer to the Verification and Validation KPA to learn more about quality assurance activities performed locally by maintainers [SEI 2002, PA145.R102].

### 7.3.3 Expected Results of this KPA

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- A quality assurance plan for maintenance exists.
- Objective evidence (such as reports) shows that audits were performed and corrective actions were undertaken.
- The use of maintenance processes is progressively becoming uniform. More and more, maintenance activities conform to planning, standards, and standardized maintenance processes.
- A more precise measurement of maintenance process execution will have been published.
- Nonconformities are identified and followed up before delivery.
- Data on the use of standardized maintenance processes are collected.
- There is objective evidence of more manager involvement and escalation of internal processes to upper management when nonconformities cannot be solved at the maintenance operational level.

## 7.4 MAINTENANCE MEASUREMENT AND ANALYSIS KPA

The Maintenance Measurement and Analysis KPA provides the measurement and analysis capability that is used to support maintenance management's information needs [SEI 2002, PA154].

The objective of this KPA is to ensure that all of software maintenance's information needs are defined and to specify what measures, analysis techniques, data collecting activities, repositories, and reports will be used by software maintenance employees, customers, and managers.

### 7.4.1 Goals of this KPA

1. Align data collection and analysis activities.
2. Establish analyses, their informational objectives, and the measures needed to perform these analyses.
3. Develop data collection processes and processes for their storage in image data repositories.
4. Communicate maintenance analysis results.

### 7.4.2 Links with Other KPAs

Refer to the Maintenance Planning KPA to learn more about estimating effort for software maintenance activities [SEI 2002, PA154.R101].

Refer to the Requests/Software Monitoring and Control KPA to learn more about the data required to control and follow up maintenance work items [SEI 2002, PA154.R102].

Refer to the Configuration and Version Management KPA to learn more about managing maintenance products (including measures) [SEI 2002, PA154.R102].

Refer to the Maintenance Process/Service Definition KPA to learn more about collecting data that lead to establishing maintenance measures [SEI 2002, PA154.R106].

Refer to the Maintenance Training KPA to learn more about training activities concerning collecting and using measures [S3<sup>™</sup>].

### 7.4.3 Expected Results of this KPA

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- The measures being used meet specific objectives and stem from documented information needs.
- Measures and analyses are benchmarked using the industry's exemplary practices.
- Data collection procedures and instructions are deployed and supported by tools.
- Useful, specific, and updated reports and analyses are available.

## 7.5 CAUSAL ANALYSIS AND PROBLEM RESOLUTION KPA

Causal analysis identifies causes of failures and other problems, and what needs to be done to avoid them in the future [SEI 2002, PA155]. The interdependence be-

tween business processes and software will cause the whole organization to become paralyzed if the software fails. Service continuity and software availability require an efficient problem resolution process.

The objective of this KPA is to identify priorities and come up with improvement analyses and recommendations. Here, the objective is to identify the causes of failure in order to eliminate them. Improving quality is also an objective.

### **7.5.1 Goals of this KPA**

1. Establish criteria for evaluating common faults and failures.
2. Improve problem resolution quality and efficiency.
3. Eliminate the causes of failure and prevent their recurrence.

### **7.5.2 Links with Other KPAs**

To learn more about analysis of process performance and quality, and discover when it falls outside acceptable limits, see the Quantitative Management road map of the Maintenance Process Performance KPA [SEI 2002, PA155.R101].

Refer to the Maintenance Innovation and Deployment KPA to learn more about choosing and deploying improvements in the maintenance organization [SEI 2002, PA155.R102].

Refer to the Maintenance Measurement and Analysis KPA to learn more about establishing software maintenance measurement and analysis objectives [SEI 2002, PA155.R103].

### **7.5.3 Expected Results of this KPA**

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- Software maintenance costs decrease.
- The handling time for a customer's MR is shorter.
- The training time for maintenance resources is shorter.
- Software complexity decreases.
- Software documentation quality increases.

## **7.6 SOFTWARE REJUVENATION, MIGRATION, AND RETIREMENT KPA**

Software rejuvenations activities are used to solve problems related to application software's decline or to address a change in its technical environment. A decision criterion is necessary to decide whether or not this activity is economically benefi-

cial for the software [Koskinen 2005a]. Attempts to increase its quality, efficiency, and general availability will thus be made. In some cases, attempts to decrease maintenance costs and shorten waiting times will also be made. This road map includes migration of software to another technical environment (platform). These are considered preventive maintenance activities. Components of the software must be explored to try and document it, obtain additional information, or change its structure to make it easier to understand and maintain.

The objective of this KPA is to use software rejuvenation techniques when their need has been confirmed and approved by maintenance planning: (a) redocumenting the software, (b) restructuring the software, or (c) performing software retro-engineering.

### **7.6.1 Goals of this KPA**

1. Decrease the cost of software maintenance.
2. Shorten waiting times for service requests.
3. Increase the maintainer's ability (level of service, knowledge, and comfort) to evolve software.
4. Decrease software complexity.
5. Update, complete and standardize software documentation.
6. Create functional and architectural documentation from the software's source code and data.

### **7.6.2 Links with Other KPAs**

Refer to the Maintenance Planning KPA to learn more about planning that serves as a starting point for software rejuvenation activities [S3<sup>m®</sup>].

Refer to the Maintenance Process/Service Definition KPA to learn more about the need to define software rejuvenation processes. Refer also to the process documentation and standardization [S3<sup>m®</sup>].

Refer to the Maintenance Training KPA to learn more about software rejuvenation training activities [S3<sup>m®</sup>].

### **7.6.3 Expected Results of this KPA**

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- Software maintenance costs are reduced.
- Maintenance employee morale is increased.
- The service time for MRs by customers is shorter.
- The training time for maintenance resources is shorter.

- Software complexity decreases.
- Software quality (e.g., documentation and structure) increases.

## 7.7 SUMMARY

In this chapter, the last process domain of S3<sup>m®</sup> was presented, including its relationship with other process domains. The goals were stated, as were the expected results of implementing the processes associated with this domain.

## 7.8 EXERCISES

1. What is the fundamental difference between a safety copy and a configuration image?
2. What is the importance of configuration management in software maintenance? How does it differ from software development?
3. At what level of maturity are you if maintenance employees are working on their own and it is only when a problem occurs (often signaled by an outside stakeholder or a customer) that the line manager does a quick review of the problem or nonconformity? What is required of you at level 3?
4. Similarly, at what maturity level are you when analyses are carried out mainly after a failure or a problem that caught the organization off guard, and what is required of you at level 3?
5. How can the level 3 analyses support your service agreement?
6. At what maturity level can a structured resolution process be seen?
7. Which practice(s) can help you classify maintenance defects?
8. Explain the refactoring technique that does not require a specific improvement project.
9. For what reasons should the maintainer conduct studies and perform cost–benefit analyses for reengineering projects?
10. Describe the various techniques of software reengineering.

# Chapter 8

## Exemplary Practices— Process Management

### This chapter covers:

Process Domain	Key Process Area	Road Maps
Process management	Maintenance process focus	<ul style="list-style-type: none"><li>• Responsibility and communication</li><li>• Information gathering</li><li>• Findings</li><li>• Action plan</li></ul>
	Maintenance process/service definition	<ul style="list-style-type: none"><li>• Documentation and standardization of processes/services</li><li>• Process/services adaptation</li><li>• Communication of processes/services</li><li>• Process/service adaptation</li><li>• Repository of processes/services</li></ul>
	Maintenance training	<ul style="list-style-type: none"><li>• Requirements, plans, and resources</li><li>• Personal training</li><li>• Training of new personnel</li><li>• Training for projects predelivery and transition</li><li>• User training</li></ul>
	Maintenance process performance	<ul style="list-style-type: none"><li>• Definition of maintenance measures</li><li>• Identification of baselines</li><li>• Quantitative management</li><li>• Prediction models</li></ul>
	Maintenance innovation and deployment	<ul style="list-style-type: none"><li>• Research of innovations/improvements</li><li>• Analysis of innovations/improvement proposals</li><li>• Piloting selected innovations/improvement proposals</li><li>• Deployment of innovations/improvements</li><li>• Measurement of innovations/improvement benefits</li></ul>

## 8.1 MAINTENANCE PROCESS FOCUS—DETAILED EXEMPLARY PRACTICES

### B.1 Level 0

Pro1.0.1 The software maintenance organization does not carry out structured process improvement activities leading to persistent and controlled process improvements [S3<sup>m®</sup>].

The organization does not have a planning process, a quality improvement process, or a software maintenance life-cycle methodology. Neither IS/IT management nor software maintenance middle managers acknowledge the need for, or value of, a quality program. The customers, end users, maintenance services, and software products are never approached with a view to identifying quality improvement opportunities [IT Governance Institute 2007, AI7].

### B.1 Level 1

**Foreword.** At maturity level 1, management begins to be aware of the need for quality and improvement. Some individuals carry out improvement initiatives. The improvement focus is often limited to technical aspects of software maintenance. The quality of the operational processes is typically not measured. Management informally assesses quality and the candidate initiatives for improvement [IT Governance Institute 2007, AI7].

Pro1.1.1 Within the software maintenance organization, improvement is carried out in an informal way [S3<sup>m®</sup>].

At this maturity level, it is recognized that improvement is carried out by individuals and the impact is limited. No time, budget, or projects have been defined to improve maintenance processes and products.

Pro1.1.2 Some individual improvement initiatives aim mostly at improving technical aspects of software maintenance processes [S3<sup>m®</sup>].

It can also be observed that, since improvement relies on individual initiatives, improvements are mostly carried out on technical aspects of the software maintenance. Examples are improvements in naming conventions of libraries, technical tools, and documentation, and compiling scripts and data cleanup activities.

## B.1 Level 2

**Foreword.** At maturity level 2, the improvement activities are carried out locally (that is, each software maintenance organization improves independently from the others). There is no coordination with corporate initiatives. A software maintenance improvement coordinator typically supports the local process improvement activities as an additional task. The IS/IT organization may have an improvement program, but the software maintenance organization does not link to it. General quality training (not specific to maintenance, i.e., Malcolm Baldrige, ISO 9001, and CMMi) is sponsored by IS/IT management [IT Governance Institute 2007, A17]. There are different opinions and lists of needed improvements, and software maintenance managers have a personal list of improvements completed and planned for their section (within the current budget cycle). A minimal set of measures is collected [IT Governance Institute 2007, A17]. Studies and pilot projects of process improvement activities are discussed and considered. Some maintenance organizations compare their processes to try to harmonize and improve them. There is still no significant budget dedicated to maintenance process improvement. Typically, the maintenance managers discuss process improvement initiatives and report on progress within the context their annual performance review with their respective managers.

Pro1.2.1 A quality and process improvement program has been initiated at the IS/IT organizational level. The software maintenance managers are aware of this program and have had initial quality awareness training [Zitouni 1996, MCO12/01, S3<sup>m®</sup>].

An improvement program must be initiated by the organization that owns the software maintenance function. To meet this exemplary practice, the maintenance managers must have been exposed to the organization's improvement program objectives. This exemplary practice does not imply that the improvement program has reached the software maintenance organization yet.

The improvement activities are beginning to be coordinated across the whole IS/IT organization. The software maintenance managers have, therefore, received the basic quality awareness training required to understand the organizational approach to improvement and the planned deployment of improvement activities. This training was provided by the management of IS/IT or of the owning organization. The training for managers typically covers the following topics:

- Key concepts, models, and methods for process improvement
- Mapping to various models (i.e., CMMi, ISO 9001, Malcolm Baldrige, ITIL)
- The organization's improvement goals
- Planning, actors, and support staff
- Deployment schedule and targeted sites within the next one-to-three-year time frame

***Responsibility and Communication Road Map***

Pro1.2.2 A representative of the software maintenance organization is assigned to plan and coordinate improvement activities [S3<sup>m®</sup>; ITIL 2007e, chapters 6 and 8].

A maintenance resource is assigned on a full- or part-time basis. Initially, this improvement responsibility is assigned to a member of the software maintenance organization (that is, a senior maintainer well respected for his knowledge and his achievements in software maintenance). This maintainer carries out the following duties:

- Alignment with the process improvement activities of the other organizational units (IT management, developers, infrastructure and operations, help desk)
- Selection of an improvement model as a baseline for comparing the organization's processes; for example: S3<sup>m®</sup>, CMMi, ISO 9001
- Documentation of maintenance employees' responsibilities when they participate in the processes (or interfaces) of the other organization's units in terms of:
  - a) Interfaced processes
  - b) Products and processes currently in use in their environment
  - c) Products and processes currently used by the customers/end users
- Information collection as well as communication concerning the goals, priorities, and potential areas of software maintenance improvement
- Promotion of process improvement to software maintenance employees and managers
- Review of processes, identification of improvement opportunities, and suggestion of improvement actions and related goals in terms of:
  - a) Productivity
  - b) Default reduction
  - c) Cost reduction
  - d) Delay reductions
  - e) Employees' motivation
  - f) Customer satisfaction
- Development of an annual set of prioritized activities for software maintenance improvement
- Coordination of the activities and processes for software maintenance improvement

***Information-Gathering Road Map***

Pro1.2.3 Results of the software maintenance products/services customer survey is used to identify candidate improvements [S3<sup>m®</sup>; ITIL 2007b, 4.2.5.4].

The software maintenance senior manager ensures that the organization carries out, once a year (or more frequently), a survey with a meaningful sample of customer/user/interface groups. The aim is to gain a better understanding of whether or not the processes and products meet their expectations and quality/service objectives. This survey should cover the following topics:

- Professionalism and politeness of the maintenance work force
- Perceived competency and skills of the maintenance work force
- Types and quality levels of the products and services delivered
- Waiting time and response time in processing and closing requests/events
- Evaluation of the application software portfolios currently in use
- Information status of the tracking of requests in progress
- Flexibility and alternative solutions in request-priority shuffling
- Failure rate of application software and recovery time after failure
- Representation to, and negotiation with, subcontractors
- Services costs, quality, and value

**Pro1.2.4** The observations, comments, and complaints from users/customers and interface groups (developers, operations, help desk, subcontractors, etc.) are used to collect data for identifying candidate improvements [S3<sup>m®</sup>; ITIL 2007e, 4.1, 3.2].

A process for collecting observations, comments, and complaints is implemented. During their numerous daily contacts, the maintenance engineers collect and record, in a consistent manner, information that will be useful for process improvements. This information is made available to the process improvement coordinator, and taken into account in planning improvement activities.

**Pro1.2.5** Comparisons of application software profiles and relevant internal benchmarks are used to identify candidate improvements [S3<sup>m®</sup>].

Internal benchmarking is a comparison technique for self-improvement purposes. The literature [Abran 1993b] describes how the maintenance organizations can use comparisons internally, among themselves, as a way to identify the internal exemplary practices.

This approach can be applied relatively easily, and allows for in-house harmonization with what is done elsewhere in the industry before an external benchmark is established [ITIL 2007e; S3<sup>m®</sup>].

**Pro1.2.6** The data on software failures is collected and used to identify candidate improvements to maintenance processes/products and also to the many interfaces with the other interface groups (developers, operations, help desk, subcontractors, etc.) [S3<sup>m®</sup>].

Software failure can occur for various reasons that are not necessarily easy to identify and understand at first. Collecting data on failures and conducting analysis of their causes is a way to identify possible improvements, first locally within the maintenance organization and, subsequently, across its many interfaces (i.e., the problem-resolution process, which involves help desk, computer operations, and other suppliers). Services, as perceived by the end user, are delivered through several IS/IT organizations (developers, operations, help desk, subcontractors, etc.). Each IS/IT support organization must ensure that its process and its interface, in the service chain, are coordinated [April 2001].

Pro1.2.7 The maintenance organization is subject to internal audits (from internal auditors (e.g., COBIT, ISO 90003 4.2.4 e or other types of audits) and the results are used to identify candidate improvements [S3<sup>m®</sup>].

Software maintenance senior and middle managers accept and have a positive attitude to internal and external audits when their organization is subjected to them. Audits provide an external view of the software maintenance processes/services, products, and controls. Audit reports must be available to the process improvement coordinator, and taken into account when planning potential improvement activities.

Pro1.2.8 A maturity assessment of some processes has been performed. At least one maintenance organization has conducted a process maturity assessment and the results are used to identify candidate improvements [Camélia 1994, 1.4.2.1; Zitouni 1996; MCO.12/02; ITIL 2007e, 5.3.7; S3<sup>m®</sup>]

The strengths and weaknesses of the software maintenance process are evaluated against a reference model. A process assessment (e.g., SCAMPI [SEI 2000]) has been carried out, which, at this level, can be as a pilot project or an initial assessment. This activity allows the usefulness of the chosen improvement model to be verified; it also allows maintenance personnel (management, improvement coordinator, and some maintenance engineers) to be trained in software engineering improvement process activities in general. This improvement activity is led by software maintenance engineers. It allows the maintainers to better understand the software process improvement activities. The results are used to identify candidate improvements.

### **Findings Road Map**

Pro1.2.9 A list of candidate improvements, ordered by priorities, is developed and used as a guide for developing the software maintenance improvement program [S3<sup>m®</sup>].

Using the data collected about potential improvements, the software maintenance improvement coordinator produces, communicates, and discusses with the managers a list of candidate improvements, which need to be ordered by priorities. This list includes activities to be completed, in progress, and planned to begin within the current annual budget. This document becomes the software maintenance improvement program. At this point, it is not fully aligned with the IS/IT improvement program.

### **Action Plan Road Map**

Pro1.2.10 The list of the top 10 possible improvements is discussed, and actions plans are drawn up by middle and senior management. The planned improvement activities/projects are funded within the annual budgets and current operations [Zitouni 1996 MCO.12/03; ITIL 2007e, 4.4; S3<sup>m®</sup>].

Improvement projects are identified and prioritized by software maintenance senior management. The improvement activities are performed, in addition to the current work, using current resource allocations; that is, no “significant” additional funds are freed up for improvement activities. These improvement activities are in addition to the current software maintenance operational workload.

Pro1.2.11 Improvements to some processes have been initiated. The annual plan, of each maintenance organization, includes both the improvement activities planned and carried out during the year [Camélia 1994, 3.3.2.2; S3<sup>m®</sup>].

Each software maintenance middle manager is responsible for conducting or expected to conduct, in his (yearly) work plan, some improvement activities. This work plan reflects the achievement on process improvements with respect to the goals and priorities. The yearly evaluation of a maintenance manager includes criteria about progress in his/her operations improvement activities.

## **8.2 MAINTENANCE PROCESS/SERVICE DEFINITION—DETAILED EXEMPLARY PRACTICES**

### **B.2 Level 0**

Pro2.0.1 The software maintenance organization does not carry out process/services definition activities. It does not provide a list or directory of services to its customers [S3<sup>m®</sup>].

The software maintenance organization is not yet sufficiently clearly defined to focus on software maintenance goals, nor is there yet a clear recognition of the need

to establish policies, procedures, and processes specific to software maintenance. No tailored, structured processes have been implemented to tackle software maintenance activities, including related required techniques and infrastructure. The only available documentation, and related maintenance work environment, has been provided to developers, suppliers, and subcontractors [IT Governance Institute 2007, AI4].

## B.2 Level 1

**Foreword.** At level 1, the software maintenance managers are aware of the need for processes, but little is actually done. Maintenance processes are occasionally defined, but rarely deployed across all maintenance organizations. Maintenance has little impact on software activities and is only available in some organizational units. A large part of the maintenance documentation and procedures are not up to date, and there is almost no integration across procedures between the various IS/IT supporting organizational units at the interface [IT Governance Institute 2007, AI4].

Pro2.1.1 The maintenance processes/services are informal and based on the experience of individuals [S3<sup>m®</sup>].

A reactive approach exists with respect to processes. Processes are developed when a problem has been identified/reported and it persists. There is not yet a precise approach for developing and deploying maintenance processes.

Pro2.1.2 Individual initiatives, for defining maintenance processes/services, mainly address technical aspects of the software or describe, in a local format, the activities of a specific maintenance organization [S3<sup>m®</sup>].

Some employees have personal notes and documents about the processes and software products for which they are responsible.

## B.2 Level 2

**Foreword.** At level 2, the process definition activities are performed by each maintenance organization independently, and no clear description exists of the processes at its interface (i.e., customers, end-users, developers, operations, and suppliers). Approaches are used for development of processes and procedures, but without intergroup coordination, a structured approach, or a framework [IT Governance Institute 2007, AI4]. IS/IT or other organizational units of the organization may have process definition activities, but software maintenance is not involved in a corporate initiative. The processes that have an impact on customers

are documented, but there is no uniform approach; therefore, their accuracy and availability rely greatly on individuals rather than on a formal process. Some training material is available, but this is piecemeal and its quality depends on the individuals involved. The procedures and quality of software maintenance activities and products varies from poor to very good, with little homogeneity or integration across the various maintenance organizations [IT Governance Institute 2007, AI4].

### ***Documentation and Standardization of Processes/Services Road Map***

Pro2.2.1 There is at least one description of maintenance processes/services that is used by the customer and offered by the maintenance organization [ISO 90003, 4.1a and 7.2].

One of the software maintenance organizations has defined and documented standardized processes describing their activities. These are shared and used with the customers.

Pro2.2.2 Portions of the maintenance processes/services/resources are documented and deployed in the software maintenance organization [Camélia 1994 (a1, a4), ISO 90003, 5.4.2].

Documentation exists to describe some maintenance processes, services, and resources (personnel, tools, software, and environments). This documentation is:

- Up to date
- Used by the maintenance engineers in their daily work

A limited set of techniques (for maintenance programmers and maintenance analysts) is standardized and documented [Camélia 1994, 7.1.2.6]; for instance, programming standards and naming conventions for the evolution of particular software are documented in a programmer's guide for the maintenance unit.

Pro2.2.3 There are activities under way, at the organizational unit level, to document and standardize software maintenance processes/services [Camélia 1994, 6.1 and 6.2; ITIL 2007e, chapter 3].

The necessity for standardization of software maintenance processes has been recognized by the organization. Some standardization activities have been initiated, but not necessarily in all software maintenance organizations.

Pro2.2.4 Software maintenance management acknowledges and promotes the use of standard processes/services and the use of external standards relevant to software maintenance (i.e., ISO 9001, ISO 12207, ISO 14764) [S3<sup>m®</sup>, ITIL 2207e, 3.11].

The usefulness and necessity of using standards for improving software maintenance has been acknowledged. This translates into acceptance, by managers and engineers, that software maintenance is a specific domain and that it requires its specific specialized terminology and related standards to better communicate with other software engineering organizations, suppliers, subcontractors, and out-sourcers. This exemplary practice promotes the introduction of a standard terminology and standardized processes, activities, and categories of work based on recognized international standards (open), rather than terms defined in-house and issued from within (closed).

## 8.3 MAINTENANCE TRAINING—DETAILED EXEMPLARY PRACTICES

### B.3 Level 0

Pro3.0.1 Software maintenance organizations do not have structured training activities that address predelivery and transition, the software maintained and its technical environment, the maintenance processes/services or the engineers' morale/career plans [S3<sup>m®</sup>].

What also characterises level 0 is the lack of human resource management in terms of training plans, training activities, and career management. There is little awareness on the part of IS/IT management of the importance of aligning the training activities of the maintainers with the project plans of new software projects. There is no individual or group responsible for training [IT Governance Institute 2007, PO7]. The status of a maintainer is lower than that of a developer. After a while, the maintenance engineer feels let down by the organization. Maintenance engineers feel they should move on or accept this lower status in the organization. At this level of maturity, there is no career management and the impact is perceived in the morale of the work force.

### B.3 Level 1

**Foreword.** At this maturity level, the organization still does not have a lot of information about what is needed in terms of training for software maintenance personnel. Kajko-Mattsson and coworkers [Kajko-Mattsson et al. 2001] note that the software maintenance engineer does not have much opportunity to develop his professional skills, except for his own personal interest. The typical scenario, at this

maturity level, is that education and training is offered in only in the following two situations: the arrival of a new employee or the transitioning of new software. Management is aware of the need for training plans, but has not taken action yet. The training process is informal and reactive to emergencies [IT Governance Institute 2007, PO7, ITIL 2007d, Table 3.1].

**Pro3.1.1** Technical training in the software maintenance organization is provided when there is an urgent need [ISO 12207; ISO 90003, 6.2.1; Kajko-Mattsson et al. 2001b, ME-1.2].

Periodically, new software is transitioned to software maintenance and must be maintained. Similarly, new platforms, upgraded infrastructure, and new processes are introduced in software maintenance. When these changes are introduced, technical training is provided on a just-in-time basis to handle the situation.

**Pro3.1.2** Individual training initiatives are aimed mainly at the technical aspects of software maintenance and the products they support. A pairing process appears where a senior staff member is assigned to a junior employee as a mentor and to answer his or her questions [S3<sup>m®</sup>].

Individual training is geared toward the understanding of particular software and its the technical processes for maintaining the infrastructures that are involved. Junior engineers looking for help on a specific task will direct their questions to the senior engineers. In software maintenance, pairing ensures that at least one individual can be available at any time.

**Pro3.1.3** The training plans for new engineers address generic topics about management, processes, and software maintenance activities [ISO 12207; SEI 2002; PA158.IG101.SP101.W101; ISO 90003 5.4.2; Kajko-Mattsson et al. 2001b, ME-1.1].

At this level of maturity, the new employee will be provided with a brief overview on the organizational unit processes. Training will also address the maintenance request management system and source code and documentation libraries, and compile rules and procedures with operations. Training needs are identified and documented in the employee training plans for the year.

## **B.3 Level 2**

**Foreword.** At this maturity level, the training process includes the notion of education. It is defined, documented and used locally; personnel are trained for local and immediate needs by the organizational unit [IT Governance Institute 2007, PO7].

Kajko-Mattsson [Kajko-Mattsson et al. 2001b] notes that at this level an effort is made to meet the basic criteria for the education and training of software maintenance engineers. The organization has an increased awareness of the need to better manage these individuals.

Pro3.2.1 Maintenance engineers become proficient and periodically update their proficiency with the software they maintain and its infrastructure [Kajko-Mattsson et al. 2001, ME-2.3; Kajko-Mattsson 2001b, ME-Resource-5; ISO 90003 6.3].

It is necessary that the engineers assigned to software maintenance acquire a general knowledge of the architecture and functionality of the software, as well as a detailed knowledge of its internal components and processing of data elements. This knowledge level will impact the productivity of the maintainers in daily tasks [Kajko-Mattsson 2001b, p. 615]. Recent studies refer to the lack of general knowledge about the software as a cause of maintenance problems [Kajko-Mattsson 2000]. The maintainers, because they most often work on small portions of the software maintained, do not always have the opportunity to acquire a global view of complex software. It is then necessary for the maintainers to reacquaint themselves with the software on a regular basis. This also means visiting the end users to learn how the functionality of the software is used. Training periods must be planned and allocated to individuals. The “People CMM” [Curtis 1995] recommends such activity to improve knowledge about the architecture of particular software and its specific domains.

Pro3.2.2 The maintenance engineer is trained and motivated to perform well when using the processes/services and their support role [ISO 12207; Camélia 1994, 5.2 and 5.3; SEI 2002, PA158.IG102; ISO 90003 6.2.2; Kajko-Mattsson et al. 2001b, ME-2.4; Niessink et al. 2005, 5 5.2 ability 5; ITIL 2007b, 9.2.5].

This exemplary practice is aimed at addressing the motivation issues of software maintenance engineers in order for them to become well informed about their processes and how they carry out their tasks. It is also aimed at encouraging the engineers to come forward with improvement recommendations and to become personally involved in process improvement. The individual must be motivated to carry out his work in a disciplined manner. All the goals behind any process must be explained. Maintenance managers and engineers must do the extra work to document the maintenance processes well. Studies by Dr. Kajko-Mattson show that, even if the process is well documented, it is rarely executed as prescribed [Kajko-Mattsson 2001b]. Implementing this exemplary practice requires the maintenance organizations to:

- Identify where there is a lack of training
- Identify weaknesses in current training
- Review documents for training on processes
- Assess knowledge about maintenance processes
- Invite engineers to formulate suggestions for improvement of training on processes

Pro3.2.3 Training on communication with customers is offered to software maintenance engineers [Kajko-Mattsson et al. 2001b, ME-2.8; Niessink et al. 2005, 5.2 ability 4; ITIL 2007b, 6.2.4.3].

The maintainer works directly with customers. It is, therefore, necessary to offer training on verbal communications and on interpersonal communications.

Pro3.2.4 Use of internal benchmarking data to guide the training of maintenance resources [S3<sup>m®</sup>].

Internal benchmarking is a comparison technique for improvement. Some authors [Abran 1993b] describe how maintenance organizations can use comparisons with one another to better understand the similarities and differences, and also to identify the internal exemplary practices of the organization. To demonstrate the deployment of this exemplary practice, maintenance managers will need to have carried out a comparison of the investment in training in the other software specialties and other organizational units. Such a comparison can be illustrated graphically, and must be used in discussions on training needs with managers and with the training group.

### **Requirements, Plans, and Resources Road Map**

Pro3.2.5 Financial resources are available to maintenance managers for the education and training of each maintenance engineer [Kajko-Mattsson et al. 2001b, ME-2.1].

The maintenance organization has a training budget. This budget is based on individual needs and is allocated according to needs. Documentation is available that describes:

- Successfully completed training courses
- Internal training activities followed by employees
- Plans for future training

Pro3.2.6 There are plans and records that describe the education and training needed for each maintenance engineering position and application software. This training plan documents the training needs, the courses offered, and related material, credits, resources available, and the schedule of education and training activities [Niessink et al. 2005, 6.7, activity 1; SEI 2002, PA158.IG102.SP102; Camélia 1994, 5.2 (a1, a2)].

Keeping up to date is an engineering responsibility. An education and training plan must cover the following topics: ethics and professionalism, continuing training, external standards and internal maintenance standards and processes, processes with interfacing organizations, technical infrastructure, the application software and its related business rules, and data structures.

### ***Personal Training Road Map***

Pro3.2.7 Training time is planned [SEI 2002, PA158.EL102; Kajko-Mattsson et al. 2001b, me-2.2].

At this maturity level, it can be observed that the engineer takes roughly 2 hours per week to familiarize himself/herself with exemplary practices that contribute to improving work efficiency. The engineer can also attend or give internal presentations to colleagues during this time [Humphrey 2000].

### ***Training of New Personnel Road Map***

Pro3.2.8 Senior maintainers familiarize new employees and prospective maintainers with the maintenance of the software for which they are responsible [S3m®].

Senior maintenance engineers must familiarize the new engineers with processes, activities, infrastructure, and techniques for the maintenance of specific software. This activity address the following topics:

- Documentation and libraries for code and object sources
- Editors, tools for comparing programs and data files, compilers, assemblers, linkers, operating software loaders, debuggers, simulators, emulators, static code source analyzers, test tools, documentation tools, and configuration management and database management systems
- Operations and support hours of operational support

**Training for Projects in Predelivery and Transition Road Map**

Pro3.2.9 For each development project to be transitioned to software maintenance, the training needs are defined for both the technical and management responsibilities (e.g., nature of training, identification of the staff to be trained, timing of training, etc.) [SEI 2002, PA166.IG102.SP104.SubP102; S3<sup>m®</sup>].

The predelivery and transition of software to an operational environment is one of the most important and most overlooked of software development tasks [Pigoski 1997]. At this maturity level, the training offered by the development project during pre-delivery of the software and on its maintainability is minimal (i.e., roughly 3 days [Pigoski 1997, p. 120]). To meet this exemplary practice, the maintainer must assess and review the initial proposal on the training of maintenance personnel by the development project team. During the development life cycle, the maintainer must set up two-way communication to formulate and clarify his need to (a) ensure a smooth transition and (b) ensure that he meets his responsibilities for the maintenance of the new software.

Pro3.2.10 People who work on the predelivery and transition of a project to maintenance receive the training deemed appropriate by the software developer [SEI 2002, PA166.IG102.SP104.SubP102; S3<sup>m®</sup>].

The developer must prepare an estimate of what is required as minimal knowledge, specialized education, and training to maintain the software after its delivery. This training must have been planned and undertaken by the maintenance engineer who will be in charge of maintaining the software, and this should occur before he or she takes over final responsibility for it. Maintenance management must ensure that this process is enforced.

**User-Training Road Map**

Pro3.2.11 The end-user training material is designed in accordance with a logically documented procedure [S3<sup>m®</sup>].

The training material must be part of the infrastructure of software maintenance processes. This training material must be designed, managed, and updated accordingly.

Pro3.2.12 The user (and some other stakeholders) receives enough education and training to allow for autonomous use of the application software [S3<sup>m®</sup>, COBIT DS7].

Deficient training will potentially generate an additional number of user requests. To prevent this from occurring, the maintainer must ensure, during predelivery and transition, or after transition, that user training is adequate and made available. The user-training documentation and records must be transferred to the maintenance organization at the end of the project. Additional training sessions should be planned, by the maintainer, if this is deemed necessary when observing the number and the type of end-user requests received.

Pro3.2.13 Users receive information and training on the maintenance and support processes for all their supported application software (i.e., interfaces with the help desk and software maintenance) [S3<sup>m®</sup>, COBIT DS7].

It is important that the maintenance organization inform its end users on the services provided and on the procedures to follow to obtain these services. Maintenance services typically vary from one application software product to another, so it is important to clarify these for each application software product maintained. In some cases, information sessions should be provided on:

- Use of the request management system
- Contact numbers
- Types of maintenance services available
- Service level as per agreements

## **8.4 MAINTENANCE PROCESS PERFORMANCE—DETAILED EXEMPLARY PRACTICES**

### **B.4 Level 0**

Pro4.0.1 The software maintenance organization does not have activities organized to measure the performance of its processes, services, products and resources [S3<sup>m®</sup>].

### **B.4 Level 1**

Pro4.1.1 Some initiatives for measuring process, services, or products are carried out by individuals who have an interest in measurement [S3<sup>m®</sup>].

Some software maintenance personnel use measures of processes, services, or products they have developed by themselves for their personal use. These data are considered personal and are not being officially shared or used to manage or improve processes, services or products.

Pro4.1.2 Some qualitative measures of maintenance processes, services, and products are collected [S3<sup>m®</sup>].

Some maintenance engineers, using their close relationship with their end users, obtain qualitative measures about their processes, services, and products. These measurements are documented informally (for instance, in e-mails) and circulate locally within the organizational unit.

## **B.4 Level 2**

**Foreword.** At this maturity level, process performance deals only with basic concerns and varies between organizational units. Performance information is typically collected by middle managers for the weekly and monthly management meetings. Some performance objectives are set and the measures are used for local improvements and short-term priorities.

### ***Definition of Maintenance Measures Road Map***

Pro4.2.1 Some key maintenance processes, services, and products have measures. They are defined and used locally [ISO 14764, 6.5; ITIL 2007e, 3.7 and 4.3].

At this maturity level, the maintenance organization has identified the basic measures that are to be used by its management and customers. Maintenance managers use the information at (a) weekly management meetings and (b) at customer meetings. The two main sources of measurement at this maturity level are typically the request management system and the time sheet repository. The typical measures found at this maturity level are mainly general ones, such as:

- Percent satisfaction based on a survey
- Time sheets to record effort by activity or resource
- Number of calls for any type of service
- Number of users
- Number of breakdowns
- Number of hours worked and extra hours worked
- Number of closed requests for the period, newly opened requests, requests carried forward from a previous period, and waiting requests
- Number of maintenance resources for each application software product
- Measurement of evolution of some processes (using a model for improvement and/or ISO 9001)

### ***Identification of Baseline Road Map***

Pro4.2.2 Measurement baselines on the quality and performance of processes, services, and products, are collected, stored, reviewed, and used with the various stakeholders (customers, users, sponsors, program manager, maintenance engineers, and interface groups). They are used for improving the current processes, services, and products [S3<sup>m®</sup>].

At this maturity level, baselines are shared with the various stakeholders to describe the operational status of the maintenance portfolio that is locally managed by the organizational unit. This data is collected and stored locally by middle managers. Only middle managers have access to the rationale and to the composition of the details.

### ***Quantitative Management Road Map***

Pro4.2.3 The maintenance organization has set some performance and quality objectives [SEI 2002, PA165.IG101.SP101; ITIL 2007e, 4.3.5 and 4.1 step 2].

By setting up objectives for the quality and performance of processes, services, and products of the maintenance portfolio under their control, middle managers can forecast expected levels of services and analyze historical data to help set up more realistic objectives. With historical data available, initial objectives can be revised [SEI 2002, PA165.IG101.SP101.N102].

At this maturity level, the maintenance organization has identified basic objectives for management purposes. Maintenance middle managers collect and present performance information at weekly management meetings and meetings with customers (for example, the availability percentage of the software, say, 96%; the percentage of customer satisfaction; or the number of overtime hours).

## **8.5 MAINTENANCE INNOVATION AND DEPLOYMENT—DETAILED EXEMPLARY PRACTICES**

**Foreword.** Changes that are too major or made too quickly can overwhelm an organization and reduce or destroy its previous training investments; this would produce an impact opposite to the expected one. By contrast, too rigid a stability can also lead to stagnation, which can erode the business position of the organization, allowing its competitors to increase their market share [SEI 2002, PA161.N105]. This KPA must be carefully implemented by maintenance managers.

### **B.5 Level 0**

Pro5.0.1 The software maintenance organization does not collect information, for improvement purposes, from its stakeholders, end users, customers, and other interface groups [S3<sup>m®</sup>].

Pro5.0.2 The software maintenance organization does not study improvement or innovation proposals [S3<sup>m®</sup>].

Pro5.0.3 The software maintenance organization does not assess the impact of an innovation or improvement on the processes, services, and products [S3<sup>m®</sup>].

## B.5 Level 1

Pro5.1.1 The maintenance organization informally evaluates the benefits of its improvement and innovation projects [S3<sup>m®</sup>].

Pro5.1.2 Individual initiatives for improvement and innovation target mainly the technical aspects of software maintenance [S3<sup>m®</sup>].

Pro5.1.3 Assessments of new processes, technologies, methodologies, and tools for maintenance are performed informally [S3<sup>m®</sup>].

## B.5 Level 2

**Foreword.** At this maturity level, maintenance organizations identify improvement and innovation proposals. These proposals are discussed and approved by middle managers and are aimed at solving problems observed and visible from outside the organizational unit (often addressing a specific complaint). Objectives, mainly qualitative, are established. The deployment of improvements is controlled to prevent negative impact on processes and on application software in production.

### ***Research of Innovation/Improvement Road Map***

Pro5.2.1 New processes, services, technologies, methodologies, and tools are identified and investigated for their potential use in software maintenance [Camélia 1994, 1.5.1.3; ITIL 2007e, 5.4.2].

A maintenance engineer is assigned to investigate improvements and innovations that could be beneficial. A recommended list of potential improvements and innovations is produced based on his/her investigations.

### ***Analysis of Innovation/Improvement Proposal Road Map***

Pro5.2.2 New processes, services, technologies, methodologies, and tools are assessed and introduced at the requests level [S3<sup>m®</sup>; Camélia 1994, 1.5.1.4].

Based on the previous investigations and recommendations, the improvement or innovation is assessed as a noncritical maintenance request to gain better knowledge of its potential.

***Deployment of Innovation/Improvement Road Map***

Pro5.2.3 Improvements to some maintenance processes, services, technologies, methodologies, and tools have been initiated, in a controlled way, locally [S3<sup>m®</sup>; Camélia 1994, 1.5.1.3; ITIL 2007e, 8.3.3].

Based on the assessment results of improvements and innovations, some are deployed in the organizational unit.

# Chapter 9

---

## Exemplary Practices— Event/Request Management Domain

### This chapter covers:

Process Domain	Key Process Area	Road Maps
Event/request management	Event/request management	<ul style="list-style-type: none"><li>• Communications and contact structure</li><li>• Management of events and service requests</li></ul>
	Maintenance planning	<ul style="list-style-type: none"><li>• Maintenance planning (1 to 3 years)</li><li>• Project predelivery and transition planning</li><li>• Disaster recovery testing planning</li><li>• Capacity planning</li><li>• Version/Release and upgrade planning</li><li>• Impact analysis (PRs and MRs)</li></ul>
	Requests/software monitoring and control	<ul style="list-style-type: none"><li>• Follow up on planned and approved activities</li><li>• Review and analyze progress</li><li>• Implement urgent changes and corrective measures</li></ul>
	SLA and supplier agreements management	<ul style="list-style-type: none"><li>• Manage customer accounts</li><li>• Establish SLAs and contracts</li><li>• Execute services as per SLAs and contracts</li><li>• Report, explain, and bill services</li></ul>

## 9.1 EVENT/REQUEST MANAGEMENT KPA—DETAILED EXEMPLARY PRACTICES

### C.1 Level 0

Req1.0.1 The software maintenance organization does not manage (record, assign, control, and measure) the events and incoming service requests [S3<sup>m®</sup>].

### C.1 Level 1

Req1.1.1 Customer requests and system events, inside the software maintenance organization, are managed informally [S3<sup>m®</sup>].

Req1.1.2 An individual approach for management of customer requests and of system events exists. It is mainly based on the personal relationship between the maintenance programmer and a resource in the customer/user organization. Customers/users interact differently based on the nature of this contact [S3<sup>m®</sup>].

### C.1 Level 2

#### ***Communications and Contact Structure Road Map***

Req1.2.1 Each customer/user and system has a documented contact point that provides maintenance services [ISO 90003 7.2.3].

#### ***Management of Event and Service Request Road Map***

Req1.2.2 Each customer request and system incident is recorded by the maintenance organization or its agent (i.e., the help desk) and generates a support request (SR), a modification request (MR), or a problem report (PR) [IEEE 1998a, 4.1; ITIL 2007e, chapter 4; Kajko-Mattson 2001, PDR].

To meet this practice, each request and event must be captured and documented. For example, a unique ID number is assigned and there is a description summary for each request.

Req1.2.3 Each customer request and event is first accepted or rejected, and then, if accepted, is assigned a service category, a priority, and a preliminary estimate of its size and scope [IEEE 1998a, 4.1.2; ITIL 2007e, chapter 4; Kajko-Mattsson 2001, PI].

A request may be rejected and sent to another organization based on the SLA terms. When accepted, requests are classified according to standardized and defined categories of service. The individual in charge of the software performs the study for trade-offs that can apply to a new request for services.

Shari Pfleeger explains that many issues arise during the assessment of a new request [Pfleeger 2001]. So the maintenance team will proceed to an assessment of the insertion of the change versus the impact on stability and availability of the software. Also, what is necessary in terms of formal processes (i.e., planning, impact analysis, tests, etc.) must be judged versus the emergency of the request and of the cost that can be charged to the customer.

Req1.2.4 The accepted modification requests are assigned, in a preliminary way, to a future version of the software [IEEE 1998a, 4.1].

Assign the request to a future version of the software.

## **9.2 MAINTENANCE PLANNING KPA—DETAILED EXEMPLARY PRACTICES**

### **C.2 Level 0**

Req2.0.1 The maintenance organization does not conduct planning activities and does not produce and publish plans [IT Governance Institute 2007, P01].

There are no annual, project (predelivery and transition) or request level (impact analysis) plans. Strategic planning is not executed on the application portfolio or on the services. Management is not aware that strategic, tactical, and operational planning of software maintenance plays an important role in supporting business objectives.

### **C.2 Level 1**

Req2.1.1 Planning in the organization for software maintenance is carried out in an informal way [IT Governance Institute 2007, P01].

There is an awareness that planning is important, but there is currently no structured planning process in the maintenance organization. Strategic planning of the software maintenance portfolio is carried out in a reactive way. Predelivery and transition plans are drawn up at the last minute, in a sporadic and inconsistent way. Planning needs and issues are occasionally discussed in management meetings, but little action is actually taken. The alignment of IS/IT development and support activities

(development, application software acquisition, infrastructure, operations, and software maintenance) is not accomplished. Support and maintenance activities in IS/IT are reactive, as opposed to guided by an organizational strategy.

Req2.1.2 Individual initiatives for planning are mainly aimed at informing customers, verbally, of the possibility of processing a specific and ad-hoc request [S3<sup>m®</sup>].

The employees plan their work, at their level, according to what they can process.

Req2.1.3 Customer requests and project predelivery and transition are processed in a reactive way, as opposed to a planned way [S3<sup>m®</sup>].

The workload varies according to the requests. The maintenance personnel cannot forecast or plan the workload.

## C.2 Level 2

**Foreword.** The planning of software maintenance is understood by management, but not clearly documented (short or long term). The planning for software maintenance is performed, but not shared with the other organizational units. An annual plan is prepared only when management asks for it. Moreover, what planning there is not clearly used by the various maintenance organizational units [IT Governance Institute 2007, PO1]. At this maturity level, the predelivery and transition plan are elaborated according to the terms prescribed by the developer (or supplier, if acquired). Maintenance has initiated the development of standardized clauses to influence and discuss predelivery and transition requirements of developers and suppliers. At the operational level, the maintenance engineer develops an impact analysis statement for each modification. The maintenance workload is still managed reactively without influencing it. The workload status is reported by using a locally developed terminology of software maintenance service categories. The service commitments, for urgent situations, are based on the ability to assign a specific individual (a “champion”).

### **Maintenance Planning (1 to 3 years) Road Map**

Req2.2.1 There is a planning policy at the organizational level. Software maintenance publishes a maintenance plan for a 1- to 3-year view. This plans includes the object, scope, goals, objectives, and deliverables, and some others which are important software maintenance planning subjects concerning services, processes, resources, and products [ISO 90003 5.4; Camélia 1994 2.2 (a1)].

At this maturity level, there is a need to meet the planning requirements of the organization that manages software maintenance (Finance, IS/IT, and CIO). The maintenance organization needs, for example, to:

- Know the planning cycle of the organization
- Present its planning according to required formats and schedules
- Ensure quality and alignment of the planning content with other organizations
- Report on, use, and update the planning during the year
- Be accountable for variations

**Req2.2.2** The maintenance plan (1 to 3 years) is elaborated and updated yearly, according to a documented procedure [ISO 90003 5.4; IEEE Std. 1058.1; Zitouni 1996, MCA2.02/01].

A process for annual planning of software management is implemented [Zitouni and Abran 1996, RBA2.02/08]. The plan for software maintenance covers, among other things, both planned and unforeseen maintenance [Camélia 1994, 9.4.2.15].

**Req2.2.3** The maintenance planning activities follow the standards of the organization and are coordinated/aligned with its key support interfaces (i.e., developers and operations) [S3<sup>m®</sup>].

To meet this exemplary practice, plans must be harmonized from the perspective of these three organizational units. To do this, internal joint activities for planning and/or intergroup communications must be established at the preliminary planning stage.

**Req2.2.4** Maintenance planning (1 to 3 years) considers the possibilities of rejuvenation of the software portfolio [S3<sup>m®</sup>].

The maintenance plan discusses what software may profit from major rejuvenation activities. To do this, a case study explaining what specific rejuvenation activity may profit the software must be presented. This study explains roughly the costs and benefits of rejuvenation, and will be shown to the user for approval during an account management session and when the annual SLAs are signed.

**Req2.2.5** The maintenance plan (1 to 3 years) considers the resource needs (personnel, infrastructure, and tools) [Camélia 1994, 2.2(a4)].

The maintenance plan includes, among other things, a section on infrastructure and tools for software engineering support.

### **Software Transition Planning Road Map**

Req2.2.6 Pre-delivery and transition planning are included and initiated during the early stages of an IS/IT project [SEI 2002, PA166.IG102.SP104; ITIL 2007c, 4.1.5.3].

To meet this exemplary practice, the maintainer must be proactive and communicate the importance of early planning of the predelivery and transition activities. There should be communication with:

- The purchasing organization (when the software involves a supplier)
- The development project managers
- The computer operations organization

Req2.2.7 Every software project stakeholder is aware of the need to plan pre-delivery and transition activities [S3<sup>m®</sup>; ILIL 2007c, 6.3.2].

To meet this exemplary practice, the maintainer must ensure that the main stakeholders have had a chance to discuss the planning of transition, trying, for example:

- To be involved in the initial planning proposed by the developer
- To invite the unit responsible for infrastructure and operations to become involved in the planning of the transition

Req2.2.8 The predelivery and transition activities are part of the developer's main project schedule [SEI 2002, PA166.IG102.SP104.W101].

To meet this practice, the maintainer must validate that the transition plan, for example:

- Is produced at the same time as other steps of the development project
- Follows the same standards (level of detail, precision, and documentation) as the development plan
- Logically derives from the development project plan
- Is harmonized with the development plan
- Is subject to modifications in accordance with the modifications of the development plan

Req2.2.9 The predelivery and transition plan describes the relationship with the other development and infrastructure deployment plans [ITIL 2007e, 6.3.2].

The three different views of planning for maintenance (annual, for a transition, or for a request) show their relationship with the plans of other stakeholders; for example:

- The development/adaptation plan of the measurement repository for new software
- Infrastructure and tools plan for the engineering support for new software
- Human resources training plan
- Infrastructures and operations plan for the new software
- Development project plans:
  - Configuration management plan
  - Quality plan
  - Test plan [unit, integration, system (functional, performance, reception, installation), and regression]

Req2.2.10 Maintenance test and support activities, techniques and tools needed for the new software are defined in the predelivery and transition plans [S3<sup>m®</sup>; ITIL 2007e, 4.4.5.4].

The transition plan must describe the elements of the infrastructures that will be necessary before taking on accountability for the new software. The following are examples of infrastructure descriptions:

- Infrastructures for measurement processes
- Start-up support of maintainer (for example: access to expert resources and ability to send requests directly to the developer)
- Technical tools for the developers (environments, compilers, test tools, problem tracking, and configuration management)

Req2.2.11 The predelivery and transition plan assesses, documents, and approves the risks linked to technical aspects, the costs (hardware, software, and licenses), human resources requirements, and final transition schedule [SEI 2002, PA166.IG102.SP104.SubP101; S3<sup>m®</sup>; Zitouni 1994, MCA4.02/01; ITIL 2007c, 8.1.5].

At this maturity level, the plan for transition, mainly under the control of the developer, must have been assessed for risk. The results of this assessment influence the planning of specific activities of the transition to reduce difficulties. The assessment is performed from the point of view of the maintainer. So it must, for example:

- Be developed according to a documented approach (there are no particularities to the process of risk analysis for maintenance and it can simply be tai-

lored as a generic approach; for example, the risk management part of the PMBOK [xxxx xxxx, 2000])

- Underline risks that can have a direct impact on customers
- Include the specific aspects having an impact on software maintainability
- Try to involve the developer and the customer to obtain a joint and balanced assessment of the situation
- Try to obtain a consensus on perceived risks (qualitative and quantitative) and include the results of the risk assessment in the transition plan

### ***Planning Disaster Recovery Testing Road Map***

Req2.2.12 Every maintained software product identifies whether or not a disaster recovery service is offered. The maintainer has identified, documented, and communicated its testing role (application recovery vs. platform recovery), testing frequency, and the limits to his responsibilities [S3<sup>m®</sup>].

Req2.2.13 The data, programs, scripts, and critical documentation are copied and stored outside the maintenance site, and this is done on a regular basis [S3<sup>m®</sup>; Camélia 1994, 9.4.2.5].

Req2.2.14 A disaster recovery service may be offered. When offered, the detailed disaster recovery plan is tested (including the business recovery and data center recovery, if available) and reviewed as per the terms indicated in the agreements [S3<sup>m®</sup>; Camélia 1994 9.4.2.6].

### ***Version/Release and Upgrade Planning Road Map***

Req2.2.16 The software versions and upgrades are produced, by the maintainer, as soon as possible [S3<sup>m®</sup>; ITIL 2007c, 4.4; ISO90003 6.8].

At this maturity level, the decision to perform a modification is mainly made by the maintainer. For the upgrades and failure corrections, the maintainer makes the changes as soon as possible. For requests for changes, the maintainer will inform the customer before introducing them. The exchanges are made between the technical manager of the software and the customer affected by the modification.

### ***Impact Analysis of a Request for Modification Road Map***

Req2.2.16 The maintenance manager dynamically assigns the request work-load [SEI 2002, GP106.N101; Kajko-Mattsson 2001, TA-Resource-3 and 4].

At this maturity level, the maintenance manager assigns the requests, dictates the approaches to solutions, and monitors the processing of requests. So he or she is in charge, for example, of:

- Maintaining contact with the customer
- Influencing the relative priorities, taking into account all the views of the application software; for example: (a) changes to infrastructure and operations, (b) projects currently in development and in transition, (c) annual cycles of production, (d) current failures, (e) contractual agreements with its suppliers/outsourcers, and (f) the other priorities of IT managers
- Stating raw impact analysis and verifying the conclusions of its programmers
- Ensuring that the other maintenance resources know the established priorities
- Ensuring that the other maintenance resources work to the establish priorities
- Ensuring that the maintenance processes are followed

Req2.2.17 The operational maintenance plan is used to track the status of each customer request and to communicate the status of the work load [SEI 2002, PA162.IG101.SP104.SubP101; ISO90003 4 5.4].

At this maturity level, the plan mainly consists of a prioritized list of maintenance requests with a summary of analysis for each. International standards recommend that this list be derived from a management system for maintenance requests. This list must be managed, controlled, and communicated to show the status of the request for software maintenance services. This list must be managed according to an established procedure.

Req2.2.18 The customer request priority is assigned in close collaboration with the end user, according to a documented procedure [S3<sup>m</sup>®; Kajko-Mattsson 2001, TA-Process-PAC-3 and 4].

To meet this practice, the list of requests must be regularly approved by the customer. This practice has several secondary goals, for example:

- The customer is informed of the priorities linked to its requests.
- The request list flows and is updated.
- The customer controls the work.
- Complaints about slowness of service for a specific request are prevented.
- Discussion can be started about alternative means to accelerate the processing of waiting requests.

Several maintenance organizations have difficulty estimating and prioritizing requests [Layzell, 1990]. According to Kajko-Mattsson [Kajko-Mattsson 2001], one

way to quickly prioritize PRs is to assign a default value to them. It then becomes possible to assign the problem to a future version, and also initially (with the help of our knowledge about mean time to repair) estimate the resolution time. Determining the version to which the request can be initially assigned can be achieved using several criteria (described by Martin and McClure [1983]). It is also possible to support this decision making by assigning a financial value to each request [Camélia 1994, 8.3.2.1]. Financial aspects can be studied and taken into account in the impact analysis. The financial study can be based on:

- A cost model
- A benefit model
- Other computational methods rules for maintenance [Camélia 1994, 8.3.2.3].

The cost model takes into account the initial costs (correction or modification) and additional costs (maintenance and operations). These costs must include all resource costs: internal or external to the organization, human, financial, material, or informational [Camélia 1994, 8.3.2.4]. The model distinguishes the benefits, as follows:

- Measurable by monetary unit
- Measurable by management indicators
- Nonmeasurable (intangible)

Financial benefits are typically expressed as recoverable or nonrecoverable costs [Camélia 1994, 8.3.2.5].

Req2.2.19 PRs are rerouted to the appropriate support group only when the current holder has documented its intervention. Intergroup relationships and processes are documented and negotiated [SEI 2002, PA163.IG102.SP106.N102].

A modification to existing software can affect several actors. To meet this exemplary practice, the authorization of a specific request includes the agreement of the affected groups (for example, operational groups, developers, help desk, infrastructure, operations, etc.). The maintenance resource cannot, at this maturity level, perform a change and then present the stakeholders with an accomplished fact.

### ***Impact Analysis (PR and MR) Road Map***

Req2.2.20 A support and modification request capacity plan is created. The plan takes into account the maintenance plan (1 to 3 years) and the individual SLAs [ITIL 2007b, 4.3].

A management process for capability is developed, taking into account the various planning agreements and SLAs. The management of capability is composed of a number of activities having as a goal the assessment of the capability of processing the demand from the customer according to available resources. To meet this practice, the maintainer must show that the demand does not exceed the capability of the resources of its organizational unit according to the SLAs.

For more information on the capacity management of services and the capacity management of resources, consult [ITIL 2007b, 4.3].

## **9.3 REQUESTS/SOFTWARE MONITORING AND CONTROL KPA—DETAILED EXEMPLARY PRACTICES**

### **C.3 Level 0**

Req3.0.1 The software maintenance organization does not track or monitor its service/product commitments [S3<sup>m®</sup>].

### **C.3 Level 1**

Req3.1.1 The software maintenance service/product tracking and monitoring is carried out informally [S3<sup>m®</sup>].

Each maintenance engineer is responsible for his/her own service commitments without management control. The software has informal service levels and these service levels are not controlled.

### **C.3 Level 2**

#### ***Follow-up of Planned and Approved Activities Road Map***

Req3.2.1 The maintenance engineer ensures that the software is monitored and that the activities for solving technical problems of production software are co-ordinated [S3<sup>m®</sup>; Niessink et al. 2005, 5.3 activity 7; ITIL 2007b, 4.3.5.4].

At this maturity level, the responsibility for monitoring and problem solving lies with the software maintenance engineer. To meet this practice, there is a need to demonstrate, for a given failure, that a structured service approach is followed across the maintenance organization. For example:

- The schedule for support (outside normal hours) is known, approved, and followed.

- The process for intergroup support is known, approved, and followed.
- Time reporting and status reporting are performed according to policy.
- The escalation process is known, approved, and followed.

Req3.2.2 The commitments of various stakeholders, as agreed upon in the maintenance plans, are tracked on a periodic basis [SEI 2002, PA162.IG101.SP107; Niessink et al. 2005, 5.1 activity 6].

To meet this practice, there is a need to regularly track the progress of work against commitments with the customers and end users [SEI 2002, PA162.IG101.SP102].

See the Req2.3.1, Req2.3.2, Req2.3.12, and Req2.3.18 practices from the Planning for Maintenance KPA for more information on the commitments of, and the authorization for, maintenance planning [SEI 2002, PA162.IG101.SP107.R101].

Commitments must be subject to tracking and control, to ensure that they are met [SEI 2002, PA162.IG101.SP107.N101]:

- Review with the maintenance manager the most important/risky request commitments.
- Reexamine periodically the status of each request commitment and its progress.
- Identify and document proactively the differences, problems, and significant questions/impacts concerning commitments that may not be met.
- Document the results from reviewing commitments with customers and end users.

Req3.2.3 A review of each production software service level, monitored by maintenance engineers, is presented at the weekly maintenance management meeting [ITIL 2007b, 4.2.5.7; Niessink et al. 2005, 5.3 activity 7 and 14].

The internal SLAs with subcontractors define the precise goals for performance. The agreements must be watched. To meet this practice, levels of service must be tracked against the SLAs accepted by the customers. For example:

- There is assurance that the in-progress requests meet all the expectations of customers.
- The values for availability, for number and duration of failures, for calls for support outside normal working hours, and for the processed requests and those waiting are discussed.
- The cases that will not meet the SLAs and those that risk not meeting our commitments are identified.
- The results of this review at the level of services of software are documented.

***Review and Analyze Progress Road Map***

Req3.2.4 External and internal maintenance commitments are managed and reviewed periodically with software maintenance management [SEI 2002, PA163.IG103.SP102.SubP103; Niessink et al. 2005, 5.3 activity 8].

At this maturity level, the internal and external maintenance commitments are documented and tracked by the software engineer. The resolution process is subject to informal reviews.

To meet this practice, there is a need to obtain, on request, the status of maintenance work in progress for any or all services: user requests, support requests, pre-delivery and transition, monitoring, upgrades, disaster recovery testing, and so on, versus plans and commitments. The status of an individual work item must not be significantly different during the discussion between the maintenance middle manager, the maintenance engineer, and the customer or end user.

***Urgent Changes and Corrective Measures Road Map***

Req3.2.5 The transition plan, including its schedule, is subject to tracking and corrective actions, which are performed if needed [SEI 2002, PA162.N101; ISO 90003 5.4.3].

At this maturity level, the predelivery and transition activity schedule (mainly under the control of the developer or supplier) is subject to tracking by the maintenance engineer assigned. To meet this exemplary practice, the maintenance engineer must, for example:

- Retain a copy of the current predelivery and transition schedule (this copy is up to date and reflects the activities of the developer and of the maintainer).
- Review, weekly/monthly, the progress of activities that are his/her responsibility.
- Participate in a monthly project meeting to document this process progress (late, on track, or early).
- Try to negotiate an action plan (mutually agreed upon) for actions and remedies of identified issues.
- Ask for adjustments to, and updates of, the predelivery and transition activities.

Req3.2.6 The maintenance commitments are tracked and renegotiated, if needed [ISO 90003 5.4.3].

To meet this practice, there is a need to ensure that the stakeholders from IS/IT and the customers are involved proactively when a change to a commitment is required. Maintainers must ensure that an agreement is reached before changing a commitment.

See the Req2.2.20 practice from the Maintenance Planning KPA for more information on the need to reach an agreement with stakeholders before modifying a commitment and communicating the change.

## **9.4 SLA AND SUPPLIER AGREEMENTS MANAGEMENT KPA—DETAILED EXEMPLARY PRACTICES**

### **C.4 Level 0**

Req4.1.1 Software maintenance management has not yet recognized the need for formal contracts and SLAs [S3<sup>m®</sup>].

Responsibilities for establishing and monitoring service levels are not assigned. There are no policies or procedures for establishing contracts and SLAs with suppliers and customers. There is no measure of customer satisfaction with respect to the supplier or maintenance services. Maintenance managers do not conduct surveys of the satisfaction of their suppliers. Without data, maintenance management personnel have no service quality indicators [IT Governance Institute 2007, DS1].

### **C.4 Level 1**

Req4.1.1 The agreements with clients, subcontractors, and outsourcers are elaborated from templates, documents, and contracts from suppliers and subcontractors [S3<sup>m®</sup>].

At this maturity level, there is awareness of the need to manage the service levels and the supplier contracts, but the process is informal and reactive [IT Governance Institute 2007, DS1 and DS2]. The responsibility for control of service execution is informal. The measures are typically qualitative with ambiguous objectives. The performance reports are rare, inconsistent, and not used for decision making. The agreements are not negotiated, but rather are unilaterally proposed by customers, suppliers or subcontractors. Suppliers have the most say on agreement content. Raffoul reports that 25% of IS/IT organizations are in this category [Raffoul 2002]. This situation often results in an unbalanced relationship, characterized by loosely defined agreements, a lack of alignment in mutual objectives, few control meetings or reports on service levels, nonexistent processes, and little foundation on which to build trust and generate added value. The typical strategy for remedying customer dissatisfaction in this type of situation is to change supplier.

## C.4 Level 2

**Foreword.** At this maturity level [IT Governance Institute 2007, DS2], a consensus exists on the need to measure the service levels and the performance of suppliers/subcontractors, but performance reports are very basic and not kept up to date. They may be incomplete, not relevant, not actionable or misleading, and may depend on individual initiatives. An account manager is appointed and is in charge of negotiating contracts and SLAs. This position has defined responsibilities, but often lacks authority. The SLA and the supplier contracts are often used “as is,” without major modification or negotiation. The maintenance billing does not provide details and is often only a simulation (as in internal relationships for which detailed explanations are not requested/provided). The customer does not see the costs entailed in its request and often thinks that the maintenance service is free and unlimited. Studies by Raffoul show that 50% of IS/IT organizations are in this category [Raffoul 2002]. A number of actions are discussed to remedy the situation:

- Establish strategies to remedy the lack of communication with subcontractors and customers; define roles and responsibilities and establish objectives regarding the expectations of customers.
- Establish a management and communication structure to create a winning atmosphere.
- Establish an SLA for every service/software subcontractor to quantify the results.
- Establish a process for tracking service levels and to detect/act on issues.
- Establish a process for the management of customer accounts to process requests for new services.
- Propose a benchmarking strategy to assess and compare the competitiveness of the maintenance service and its suppliers.

### ***Customer Account Management Road Map***

Req4.2.1 Preliminary discussions are held on the introduction of a more formal SLA and contracts [ITIL 2007b, 4.2.3 & 4.2.4].

To meet this practice, there is a need to ensure the introduction of the concept of a more formal agreement on services and contracts. These discussions must promote the advantages associated with formalizing agreement/contracts, such as (a) a means for better communication, (b) a mechanism for the management of expectations, (c) a tool for reducing conflicts and for assessing performance fairly, and (d) a document that can evolve/change according to need.

Req4.2.2 A maintenance engineer is assigned as an account manager to plan and coordinate the customer account management activities [S3<sup>m®</sup>; ISO 90003 7.2.2 and 7.2.3].

A full-time assignment or a percentage of an existing function of a maintenance engineer is taken for account management. Typically, a senior and well-respected individual is chosen for his/her knowledge and interpersonal skills. The goal of this practice is to initiate and manage a business relationship with the customers and the suppliers/subcontractors:

- Using meetings to discuss the necessity for improving perceptions and for clarifying expectations
- By showing the strengths and weaknesses of the current situation
- By identifying the need for measures and for setting quality and performance objectives
- By introducing the concept of internal benchmarking (see practice Pro1.2.5) of maintenance services to sub-contractors
- By initiating discussions on SLA content and time lines to develop them

### ***Establish SLA and Contract Road Map***

Req4.2.3 The stakeholders (customers, end users, maintainers, developers, operations, and suppliers) agree on the necessity for more formal SLAs and contracts concerning software maintenance [S3<sup>m®</sup>; ISO 90003 7.5.1.g].

The result of executing practice 4.2.1 should be that all stakeholders agree that formalization of contracts and SLAs is required. If the stakeholder managers are not of this opinion, it will not be possible to meet this practice. A continuation of work on practice 4.2.1 is recommended.

Once a positive attitude prevails, a first step is to verify whether or not a formal agreement is appropriate for the business relationship. If so, the parties must agree on developing the service definitions and other content (inventory of software and existing baselines of performance and costs), as well as the processes, responsibilities, and committees for managing of the agreements/contracts. To meet this level, there is a need to ensure that the following conditions are met [Raffoul 2002]:

- The customer and the maintainer agree on the key elements of the services definition, current performance levels, and means to measure them.
- Yearly negotiation meetings are held on the change process and costs.
- A management structure is established to create and maintain constant and positive relationships. Responsibilities and processes are identified for the interface between the customer and the maintainer.

- Strategies are established to address miscommunications and badly defined roles and responsibilities.
- A process is established to address requests for services that are not part of (are excluded from) the software maintenance services.

**Req4.2.4** The maintainer selects and reassesses suppliers and subcontractors. The selection and assessment is based on assessment of their capabilities to meet the specified conditions and established criteria stated in the SLAs [SEI 2002, PA166.IG101.SP102; Niessink et al., 5.4 activity 2].

To meet this practice, the maintainer needs to follow an established procedure (often published by the purchasing or finance organization) for the selection of the subcontractors and suppliers.

**Req4.2.5** A maintenance SLA, supplier, and subcontractor contract template is defined, documented, and approved [SEI 2002, PA166.IG101.SP103; ITIL 2007b, 4.2.5.2, 4.4.1; IT Governance Institute 2007, DS1; S3<sup>m®</sup>].

With the help of the service definition practice (see Req4.2.1) and of current agreements, the parties agree on, and approve, new contracts and SLAs that are based on the adapted template. A formal agreement is treated more like a legal agreement. This agreement can be a contract, a license, or a memorandum of agreement [SEI 2002, PA166.IG101.SP103.N101]. At this maturity level, the suppliers often have a contract knowledge advantage and will often propose the initial SLA or contract terms. Often, this document, with some modifications, will be adopted. The contract is typically written and authorized by a representative of the organization [Camélia 1994, 2.2.14]. The SLAs typically reflect a technical point of view (from the supplier of maintenance service), as opposed to a customer/end user agreement. The agreement, to be negotiated, should nevertheless take into account the maintainer and the customer point of view.

### ***Execute Services as per SLA and Contract Road Map***

**Req4.2.6** The SLA concluded between the maintainer and the customer establishes the foundation of the management of the relationship [SEI 2002, PA166.IG102.SP102; ITIL 2007b, 4.2.5.3, 4.5.1; S3<sup>m®</sup>; Niessink et al. 2005, 5.4 activity 3].

Before the agreement for services or the contractual agreement is signed, the tracking capability must be set up and the current performance baselines must be documented. Activities are executed, between the customer and the maintainer (between the maintainer and its suppliers/subcontractors), as specified in the SLA [SEI 2002,

PA166.IG102.SP102]. The services are executed and there is a process for managing and tracking the agreement/contract. Tracking, meetings, and reports on the performance indicators are needed to perform this task.

Req4.2.7 The modifications to the scope of work given to the subcontractors, to modalities of the contract for subcontracting, and to the SLAs are made according to a documented procedure. Changes to commitments are reviewed and agreed between stakeholders [SEI 2002, PA166.IG102.SP102.SubP108; ITIL 2007b, 4.2.5.2, 4.5.4; S3<sup>m®</sup>; Niessink et al. 2005, 5.4 activity 6].

The agreements for services, the internal agreements, and the contracts must be updated. They must be subject to periodic changes, at least annually, to ensure that they are optimized and well aligned with the business needs of the maintainer. The agreement must be reviewed with the supplier when this is necessary [SEI 2002, PA166.IG102.SP102.SubP108].

The contractual information is formally reviewed, adapted, and coordinated, according to a documented procedure [ISO 90003 5.2.1 and 5.2.2; Camélia 1994, 2.2.15]. These reviews ensure that the services are transparent and the goals pertinent, and that nothing significant changes or annuls the agreement. If there are changes, negotiations must follow.

Req4.2.8 Formal reviews on the execution and the results of services provided are conducted at selected milestones, according to a documented procedure [ITIL 2007b, 4.2.5.7, 4.5.2; S3<sup>m®</sup>; Niessink et al. 2005, 5.4 activity 7].

Reviews and exchanges are conducted periodically between customers and subcontractors on the performance of the latter [Camélia 1994, 2.2.2.8]. The performance of the agreement is assessed and reviewed periodically by the maintainer. These meetings are held monthly, or at least four times a year. Any corrective action must be discussed when there are significant performance differences. All such actions must be written and tracked during each meeting. The files on the contractual information (for example: history data and negotiation and contract data) are kept [ISO 90003 5.2.1 and 5.2.2].

These meetings provide a forum for the presentation of explanations for failures and omissions and for making decisions on what has to be done to ensure that these situations do not recur. Also, levels of service which cannot be reached can be identified, at which time it will be possible to agree on a different level of service. Such changes can have an effect on the internal agreements of the subcontractors.

Conduct a review with the supplier, as specified in the formal agreement [SEI 2002, PA166.IG102.SP102.SubP103]. See the section on the review for more information on how to perform the tracking and manage the reviews with the suppliers.

### **Report, Explain, and Bill Services Road Map**

Req4.2.9 The maintenance organization establishes a policy for billing services [ITIL 2007b, 4.2.5.6].

This practice requires that the maintainer establish a policy for billing for his services. The need will arise for the establishment of a billing system that supports this function. This activity must be simple, just, and realistic. Billing the maintenance services is a means to:

- Force people to control their requests.
- Reduce the costs and identify the activities that do not generate many benefits.
- Allow the organization to justify its costs and the importance of the services.

Req 4.2.10 Internal customers who are bound to the services of the internal IS/IT maintainer will engage in a simulated billing [ITIL 2007a, 5.1.4.2].

To meet this practice, this group must undertake to simulate an external relationship with its customers. This transition can be achieved by following these steps:

- Step 1—bind customers and free maintenance services
- Step 2—bind customers and publish cost indicators
- Step 3—bind customers and publish real costs (internal billing)
- Step 4—free customers and charge real costs

When there are no costs, abuses of service often occur because there are no consequences. So the gradual introduction of billing is desirable. This billing can be hypothetical initially. It is not unusual for a company to subsidize the internal supplier to allow him to be competitive initially.

Req4.2.11 The maintenance organization designs, documents, and presents its bill to its many service customers (developers, operations, customers/end users) [ITIL 2007a, 5.1.3.2].

This practice requires that the maintainer make out a bill for services rendered. As a result, it will be necessary to establish a billing system that supports this function, and also to collect detailed data for billing support.

Req4.2.12 The SLA indicators are used for billing purposes [S3<sup>m®</sup>].

At this maturity level, the SLA measurement processes and performance reports are also used for billing the maintenance services, and billing can only be indicative of costs.

Req4.2.13 Raw data describing costs are available for some maintenance resources (personnel, systems, and contracts/licenses). Maintenance billing captures, presents, and explains the most important cost elements [S3<sup>m®</sup>].

It must be possible to produce a list of all the services and requests that have been processed for a defined period (monthly) [ISO 14764, p. 6]. The analysis of the work performed, by maintenance category, helps to provide a better understanding of costs.

# Chapter 10

## Exemplary Practices— Evolution Engineering Domain

<b>This chapter covers:</b>		
Process Domain	Key Process Area	Road Maps
Evolution Engineering	Predelivery and transition services	<ul style="list-style-type: none"><li>• Involvement and communication with the developer, the customer, and purchasing</li><li>• Management and control of the predelivery and transition process</li><li>• Training and knowledge transfer control</li><li>• Final transition preparation (products, environment, and problem log)</li><li>• Participation in systems and acceptance tests</li></ul>
	Operational support services	<ul style="list-style-type: none"><li>• Production software monitoring</li><li>• After-hours support</li><li>• Business rules and functional support</li><li>• Ad hoc requests/reports/services</li></ul>
	Software evolution and correction services	<ul style="list-style-type: none"><li>• Detailed design</li><li>• Evolution/Correction</li><li>• Testing (unit, integration, regression)</li><li>• Documentation</li></ul>
	Verification and validation	<ul style="list-style-type: none"><li>• Reviews</li><li>• Acceptance tests</li><li>• Installation (move to production of a change or version)</li></ul>

## 10.1 PREDELIVERY AND TRANSITION TO SOFTWARE MAINTENANCE KPA—DETAILED EXEMPLARY PRACTICES

### D.1 Level 0

Evo1.0.1 The software maintenance organization does not carry out activities of software predelivery and transition before the handover of a software [S3<sup>m®</sup>].

### D.1 Level 1

Evo1.1.1 Pre-delivery and transition, within the software maintenance organization, is carried out in an informal way [S3<sup>m®</sup>].

Periodically, the maintenance organization must take responsibility for new software that will become operational. The predelivery and transition activities are performed, to the best of his/her knowledge, by the maintenance engineer assigned to maintain the software. The most important decisions on predelivery and transition have already been made by contract or by the developer.

### D.1 Level 2

**Foreword.** The main characteristic at this maturity level is that the predelivery and transition activities be carried out according to developer's (or supplier's) terms. At this maturity level, the predelivery and transition processes are the clarification object in terms of its deliverables and its key activities.

The software maintenance organization initiates the development of standardized predelivery and transition contract clauses and processes to influence the development contracts and project plans. A checklist is used to discuss, explain, and influence (to conduct activities proposed by the maintainer) various aspects of developer predelivery and transition activities. The customers are informed of the benefits of predelivery and transition and of the transfer of responsibilities. The maintenance resources receive knowledge transfer and training, as proposed by the developer.

#### ***Involvement and Communication with the Developer, the Customer, and Purchasing Road Map***

Evo1.2.1 The software maintenance organization communicates with the developer to clarify and agree on predelivery and transition activities required for the software [S3<sup>m®</sup>; ITIL 2007c, chapter 8].

It is widely published that developers concentrate on delivery aspects as opposed to maintainability aspects of software. This exemplary practice requires that communication between the maintainer and the developer be initiated early during software development. This communication is aimed at clarifying what activities are to take place during the predelivery and transition of the software. It should prepare a smooth transition toward operations and maintenance. The following must be addressed [Pigosky 1997, p.129]:

- At what life-cycle stage will the maintenance engineer be involved during the project?
- Which intermediary products, tools, and knowledge will be transferred to the maintenance engineer?
- What criteria will be used to declare the software transitioned to maintenance?

The maintenance organization ensures that communication is established with developers early and that it participates in management committees.

Evo1.2.2 The software maintenance organization manages the relationship with the developer and the customer proactively. The objective is that there be no conflict or resistance regarding software predelivery and transition activities [S3<sup>m®</sup>; ISO 90003, 7.3.1 a8; ITIL 2007c, 4.1.5.1].

It is the maintainer's responsibility to raise predelivery and transition issues in a mature and cooperative way. Typically, the priority of the developer is to deliver the new software on time and within budget, and not necessarily predelivery and transition or software maintainability. To preserve the customer's confidence, a soft transition is of primary importance. There is a risk that management support will be lost if conflicts between developers and maintainers appear (even when these conflicts seem justified to both parties) and that these conflicts will become visible to the customer. Customer management often takes the developer's side by asking for its software as quickly as possible and within budget [Pigoski 1997, p. 83]. The customer typically has great expectations of the new software and expects continuity of service where the operational adjustments, arising during the first operational months, must be made quickly. The maintenance organization can lose its credibility with customers when the transition is not carefully managed [Pigoski 1997].

Evo1.2.3 The software maintenance organization communicates proactively with the purchasing organization (or directly with the developers) to ensure that software maintenance predelivery and transition considerations are included in the contract, as well as in the project development plan and schedule [S3<sup>m®</sup>].

The roles and responsibilities of the developer, especially in the case of external contractual agreements, describe the terms of the development project delivery and its timetable. This is, therefore, the ideal moment for the maintainer to influence the project plans in order to insert a focus on maintainability.

The maintainer has the opportunity to work with the purchasing organization (or directly with the developer), during the early stages of a development project, and to include requirements associated with the maintainability of the software and the predelivery and transition process.

To meet this exemplary practice, contractual clauses, as well as pre-delivery and transition process documents, need to be developed. Example of activities to be inserted in development project plans that identify the deliverables, the key activities, and the transition activity duration are developed. The success factors and the exit criteria associated with final transition should also be documented in this text.

Meetings to discuss, adapt, and agree on contract clauses, processes, and activities also need to take place.

Evo1.2.4 The software maintenance organization communicates with the customer to ensure that predelivery and final transition activities are well understood. In addition, this communication activity is used to introduce the basic notions of the software SLA [S3<sup>m</sup>®; ITIL 2007c, 4.1.5].

The transfer of roles and responsibilities from the developer to the maintainer, especially in the case of external contractual agreements, must be the subject of information sessions with the customers. This is the ideal moment to introduce the concepts of the SLA of this new software (before its final transition to production). To meet this practice, the maintainer must:

- Meet with the customers and explain the future predelivery and transition activities
- Explain, and show, if possible, that there is a list of outstanding problems/changes (and their assigned priorities) that will be passed on from the developer to the maintainer during final transition
- Describe the maintenance services and the number of resources assigned to maintain the new software
- Clarify the maintenance and support processes [i.e., at what time/date, and how the customer will be able to contact the maintainer for services (help desk, contact numbers, request handling, etc.)]
- Describe the general terms of the SLA

Refer to KPA Req4: *SLA and supplier agreement management* for more information on the software maintenance SLA.

### ***Management and Control of the Predelivery and Transition Process Road Map***

Evo1.2.5 The software maintenance organization develops, adapts, and uses a checklist to follow up on deliverables and key activities of the predelivery and transition of software [S3<sup>m®</sup>].

To meet this practice, the software maintenance organization needs to develop, based on the experience of several software predelivery and transition activities, a checklist describing the important components that contribute to a smooth and effective transition (refer also to practice Pro2.3.2). This checklist is used to report on progress and help structure the predelivery and transition discussions with the stakeholders.

### ***Training and Knowledge Transfer Control Road Map***

Evo1.2.6 The software maintenance engineers assigned to support the software obtain information and knowledge to facilitate the final transition of the software from development to maintenance [S3<sup>m®</sup>].

This practice requires that the transition process, its deliverables, and exit/acceptance have been previously agreed upon between the developer and the maintainer. This practice confirms that the maintenance engineer assigned to maintenance of the software understands the training and knowledge that he/she must acquire, and that activities are planned and taking place to meet the final transition deadline.

Evo1.2.7 The software maintenance organization ensures the effectiveness of maintenance training (on infrastructure, architecture, software, and documentation) provided by the developer before final transition [S3<sup>m®</sup>].

To meet this practice, it is necessary that the maintenance training agreed on for predelivery and transition has:

- Taken place successfully
- Been assessed to determine whether or not it meets the objectives of knowledge transfer
- Attained its goals

The technical training of the maintenance engineer must be evaluated by the middle manager. If deficiencies are identified, action must be taken before completion of the software predelivery and transition activities (see also training practices Pro 3.2.9 and 10).

Evo1.2.8 The software maintenance organization evaluates and discusses the efficiency and effectiveness of the user training undertaken by the customer (in terms of access, functionality, help function, and user documentation) and recommended by the developer. This must be done before the final software transition [S3<sup>m®</sup>].

To accomplish this practice, the maintenance organizational unit must be sure that user training recommended by the developer has:

- Taken place
- Met the customer's needs
- Achieved the goals established by the developer

The user's functional training must be evaluated by an intermediate maintenance manager. If deficiencies are identified, the software maintenance organization will discuss, with both developer and user, aspects of the training that will have to be carried out, following transition from the maintainer (or developer). See exemplary training practice Pro3.2.12.

### ***Final Transition Preparation (Products, Environment, and Problem Log) Road Map***

Evo1.2.9 Software maintenance engineers, who are responsible for conducting the final transition, ensure that the product documentation is obtained, that they understand its interfaces, data, and operational environment, and that they update the outstanding problems log before final transition [S3<sup>m®</sup>; ISO 90003, 7.3.1 d and 7.3.3].

At this maturity level, maintenance organizations must ensure that licensing agreements, environments, user and technical documents/products, and the outstanding problems log produced during the software development life cycle are inventoried, reviewed, delivered, and understood by the maintainer.

Evo1.2.10 Software maintenance engineers, who are responsible for conducting the final transition, ensure that the final configuration of the source code that will be used to evolve the software after final transition is obtained [S3<sup>m®</sup>; ISO 90003, 7.5.1.a3 and 7.5.1.e10].

This practice requires that the software's final configuration be revised and verified before the maintainer takes over the software. Refer to KPA Sup1: Configuration Management for more information on software configuration management. The maintenance engineer should be able, at this time, to make minor changes to the software, compile it, test it, and move it into production.

***Participation in Systems and Acceptance Tests Road Map***

Evo1.2.11 The software maintenance engineers, who are responsible for conducting the final transition, ensure that the official list of outstanding problems found during systems testing and acceptance testing are obtained. For each problem, he/she will record their categories, priority, and status in order to identify and make visible, to the customer, the extent of the request backlog that is being transferred from development to maintenance after the final transition [S3<sup>m®</sup>, ITIL 2007c, Table 4.12].

This practice asks that the maintenance engineers participate in or help with systems and acceptance testing. While the engineer is helping, there is a need to gather information and knowledge about the system and its progress toward final transition.

At the time of the acceptance testing phase of the new software, the developer must document and share the list of outstanding problems (incident reports) documented as part of the acceptance test results. This list will typically contain, for each incident report, a decision on each incident [priority, category, and status (open or closed)]. For each incident report, there must be a decision to state if it will be solved by developer or will be transferred, as is, to maintenance.

To meet this practice, each incident report that becomes the responsibility of maintenance needs to be labeled either a PR or an MR in the maintenance management software. The list of PRs and MRs must be reviewed with the customer, in order to raise awareness of the list and confirm the maintenance category and priority assigned.

**10.2 OPERATIONAL SUPPORT SERVICES KPA—DETAILED EXEMPLARY PRACTICES****D.2 Level 0**

Evo2.0.1 The maintenance organization does not provide operational support services [S3<sup>m®</sup>].

There is no operational support service to respond to questions from the users of the software. The organization did not recognize this need and this service, or does not consider this activity as a service.

**D.2 Level 1**

Evo2.1.1 The operational support service, in the software maintenance organization, is offered informally [S3<sup>m®</sup>].

The operational support service and its activities exist, but it is not possible to obtain a clear definition of it or to obtain information on its activities. These activities are not recorded on time sheets or in the maintenance management system, or clearly identified in the SLA. The responsibility (between maintenance, operations, and suppliers) for monitoring each component of operational software is not clear.

## D.2 Level 2

**Foreword.** The maintenance organization is aware of the need for an operational support service. The assistance is made available, in a basic way, by a network of well-informed individuals. There is a coherent process to handle customer information requests. The responsibility for monitoring operational software and for support outside regular business hours is clearly identified.

### ***Production Software Monitoring Road Map***

Evo2.2.1 A software operation schedule (online and batch) exists and is established on the basis of resource capability and software workload [S3<sup>m®</sup>].

To meet this practice, the maintainer must ensure that the operation schedule is known and managed by the maintenance engineer. This practice includes the batch schedule update activities, specialized script updates and updates for software for data transfers between systems. The maintenance engineer ensures that he/she is kept informed of changes and monitors the behavior of the software.

Evo2.2.2 The maintenance engineer and the operations organization are aware of their mutual responsibilities for monitoring of each piece of operational software. The maintenance engineers monitor the software proactively in conjunction with the computer operations that typically monitor infrastructure (memory, communication links, and disk space) [S3<sup>m®</sup>].

This practice requires that each support organization has identified the support and monitoring of each system component. Even if the maintainers are not in charge of supporting all aspects of the software, they sometimes have to monitor aspects that are not their direct responsibility in order to obtain a big picture (because they are often the first technical resource called in case of failure). This monitoring helps to prevent major failures and reduce them by identifying trends and alerting support staff in a proactive way. The support and monitoring responsibilities between maintainers and computer operations are clear for each piece of software. For example, it is well known what activities depend on the maintainer and what activities depend on operations, for example:

- The basic management functions of storage units, backup, defragmentation, and the clustering of storage units [Camélia 1994, 9.4.2.10]
- The basic database management functions
- The process and software testing activities for recovering from failures
- The operations-schedule-updating activities (automated calendars)

Each organization monitors the behavior of the software and its infrastructure, collaborates with and informs the other party when a problem has been identified. There is a reduction in the number of “grey areas” and “internal conflicts” about these activities. Conflict areas are identified and actions are undertaken to solve them.

**Evo2.2.3** The maintainer produces monitoring reports on the key characteristics for each service and of each piece of operational software [S3<sup>m®</sup>].

Monitoring reports are produced that describe the service level of operational software and operational support services. These reports are useful for the prevention of failures and for the identification of potential problems. The data can be printed, used for SLA management, and communicated to customers.

### ***After-Hours Support Road Map***

**Evo2.2.4** The support schedule is created, updated, and published. It is built to ensure a service coverage as agreed upon and described in the SLAs [S3<sup>m®</sup>].

A support schedule identifies the engineers on call in case of a failure outside of or during regular working hours. This schedule ensures coverage for service according to the agreement documented in the customer’s SLA. The hours are communicated and known by computer operations. The process for calling in a maintenance engineer when a failure occurs is well known.

### ***Business Rules and Functional Support Road Map***

**Evo2.2.5** The maintenance engineer has functional and business rules expertise about the software he/she maintains [S3<sup>m®</sup>].

This practice highlights the ability of the maintainer to answer questions about use (functionality) and about business rules (in the software and data elements) of the software he/she supports. The maintenance engineer can examine the source code and the data structures of the software to explain its functional behavior. The effort associated with these activities is a recognized service of software maintenance and

is publicized, documented, and measured as any other software maintenance service.

### ***Ad Hoc Requests/Reports/Services Road Map***

Evo2.2.6 The maintenance engineer offers *ad hoc* services on demand [S3<sup>m®</sup>].

At this maturity level, the maintainer is able to respond to operational requests that necessitate the elaboration of new reports and data extracts. This service can be of several types, such as:

- Helping customers to understand data or an extraction rule
- Designing and quickly executing a fast report, “one-time shot—ad hoc,” and only giving the final answer to the customer (the software developed is not part of the operational software)
- Creating a new report that will be given to the customer to run whenever he/she wants to do so
- Creating a new report that he/she will run for the user, but is not incorporated in the operational software
- Creating data extract, temporary interfaces for development teams working on other projects
- Working on computer operations projects/activities that require software application knowledge

The effort associated with these activities is a recognized part of software maintenance, and is publicized, documented, and measured like any other software maintenance service.

## **10.3 SOFTWARE EVOLUTION AND CORRECTION SERVICES KPA—DETAILED EXEMPLARY PRACTICES**

### **D.3 Level 0**

Evo3.0.1 The software maintenance organization does not provide correction and evolution services for software [S3<sup>m®</sup>].

### **D.3 Level 1**

Evo3.1.1 The software maintenance organization provides informal correction and evolution services for software [S3<sup>m®</sup>].

Each customer request is assigned to a software maintenance engineer based on the individual maintenance organization's preferences. Engineers follow their own personal/ local practices to correct or evolve software and open/close requests. Maintenance corrections and evolutions are typically performed in a reactive way. There are few formal documents and controls (it is hard to find a list of the current work load and some requests are lost) and there is no clear measurement of success.

### D.3 Level 2

**Foreword.** The detailed design, at level 2, ensures that the obligations described in the international standard for software maintenance are met. At this level, the maintainers do the detailed designs using the techniques with which they are comfortable.

#### ***Detailed Design Road Map***

Evo3.2.1 The detailed design elaborates the requirements of the user, and it is developed according to a documented procedure [ISO 12207, 5.3.6; IEEE 1219, 4.3; S3<sup>m®</sup>].

To meet this practice, a number of activities must be performed by the maintainer. For example, once the detailed design on the RMs for maintenance has identified the affected components in the software, the maintainer modifies the documentation for these components, builds test cases for verification, validates the new design (including the questions about security and safety), identifies and creates the test cases for regression, and, finally, updates the documentation with the list of modifications.

Evo3.2.2 The activities of investigation and design associated with a failure (documented in a PR) are performed according to a documented procedure [S3<sup>m®</sup>].

PRs are typically not processed like change requests. The greatest difference lies in the fact that, in a failure situation, there is no formal impact analysis followed by an authorization request submitted to the customer. In fact, the customer agrees that the maintainer is to work rapidly to repair a failure without an authorization. The maintenance engineer proceeds directly with solving the failure and will document the changes afterward.

#### ***Evolution/Correction Road Map***

Evo3.2.3 The activities of implementation (programming) are performed according to a documented procedure [S3<sup>m®</sup>].

The maintainer follows the standards of programming and naming conventions of the organizational unit (see practice Pro2.2.2) for the implementation of a modification (i.e., coding). To do this, the maintainer must follow the maintenance life cycle in stages, while taking care to document his or her results.

Evo3.2.4 The roles and responsibilities concerning operational data modifications are formally defined for the maintenance engineer [S3<sup>m®</sup>].

The maintenance engineer will have to perform data modifications (for example, to modify existing data or to destroy data) in the software that he/she supports. This operational data is the property of the users and the customers. To meet this practice, it is necessary:

- That the data owner give his/her permission before operational data are modified or destroyed
- That the stakeholders responsible for the data be informed of the events and modifications to the data
- That the maintenance engineer file the records/communications/authorizations with the request (for internal audits, external audits, and management reviews)

Evo3.2.5 The original programming languages of the software maintained are used for the corrections and evolution of the software [S3<sup>m®</sup>].

To meet this practice, the software must be corrected and evolved in the main language of the software. The complexity of the software is reduced when the number of languages is reduced to a minimum.

If this has not been done, why this language was chosen should be explained, and what the proven advantages of the use of this language are for future cost savings and maintainability stated.

Evo3.2.6 The interrelationships (traceability) between the administrative procedures, the modules, the processes, and the data are updated when the software is corrected and evolved [S3<sup>m®</sup>].

At this maturity level, the documentation and all the components involved in a modification must be identified and updated. Traceability is maintained. This practice is necessary to preserve a trace of the whole set of changes for:

- Proceeding to verification and validation
- Maintaining a measurement and a recovery (from failure) history
- Backtracking in case of future problems

- Storing all the information about a request through the various components of a software is a good way to note a change being to incorporate the unique ID of a request in all the places where a modification is performed, so that the maintainer will be able, by using retrieval software, to find the exact places where the modifications were done.

### ***Testing (Unit, Integration, and Regression) Road Map***

Evo3.2.7 The unit tests must be performed for each component that is the subject of a modification. [IEEE 1219, 4.4.2.1].

To meet this practice, the maintenance engineer must develop (a) unit test plans and (b) test data, describing all the conditions to be tested. The maintainer must make sure that the change (which has an impact on several components identified in the configuration) meets the requirements. The maintainer must also document the results of the unit tests with the objective of proving that the existing and modified components did not destabilize the software and that the modified components still meet the customer requirements [ISO 12207, 5.5.3.2].

Evo3.2.8 The integration and regression tests must be performed for each component that is the subject of a modification [S3<sup>m®</sup>].

To meet this practice, the maintenance engineer must develop (a) a plan for the integration and regression tests and (b) identify the data for the integration and regression tests, describing all the conditions to test. He/she must make sure that the components that underwent a change are integrated in the whole of the software and do not cause side effects. He/she must document the results of the tests [ISO 12207, 5.3.8].

Evo3.2.9 The documentation of the user is submitted for testing by the maintainer during the unit tests and the integration tests [S3<sup>m®</sup>].

Any documentation that will be returned to the user is subject to testing by the maintainer to make sure that it mirrors well the changes applied to the software. When documentation does not describe what the software does, a documentation fault must be signalled and the maintainer will subsequently have to modify it.

### ***Documentation Road Map***

Evo3.2.10 The internal documentation of the software is modified and kept up to date according to a documented procedure [S3<sup>m®</sup>].

The maintainer follows the recommendations of the correction and evolution guide about the need to modify the internal documentation of the software and keep it current. The maintainer must update the documentation so that it evolves with the corrections and evolutions undertaken to the software.

Evo3.2.11 The external documentation of the software is modified and kept up to date according to a documented procedure [ISO 12207, 5.3.7.3].

The external documentation is written for resource individuals who may never need to look into the source code, such as analysts and designers. This documentation is more general and explains the big picture and its interrelations, instead of a specific component. Because the software contains several components, this documentation often uses diagrams and texts explaining the interrelations. The maintainer will have to update the existing documentation. At this maturity level, the maintainer must follow basic rules with respect to [Pfleeger 2001, p. 321]:

- What problem is solved by this change and the options considered
- Describing the algorithm or the change made to the structure of data
- Documenting the modified movement of data
- Documenting the modification of the data dictionary
- Updating the documentation for the user

## 10.4 VERIFICATION AND VALIDATION KPA—DETAILED EXEMPLARY PRACTICES

### D.4 Level 0

Evo4.0.1 The software maintenance organization does not perform verification and validation activities on the maintained software [S3<sup>m®</sup>].

The maintenance organization does not plan for verification and validation activities in its maintenance methodology. Neither management nor the software maintenance middle managers recognize that verification and validation activities are needed.

### D.4 Level 1

**Foreword.** Level 1 is characterized by a chaotic process, not very different from the activity of debugging. The tests are developed reactively at the last minute, immediately after a change is programmed. There is little or no involvement of qualified personnel, specialized testing tools, or environments [Burnstein et al.

1996b; Staab 2002]. The focus of the customers, subcontractors, and developers during the development very rarely includes considerations about the maintainability tests of software or creating a test environment that will be passed on to maintainers.

Evo4.1.1 The software maintenance organization does not perform activities for verification and validation in a structured and coordinated way [S3<sup>m®</sup>].

Each maintenance engineer works independently to verify and validate his work on requests. The verification and validation work is performed in a reactive way, there are few documents, activities, or follow-up (there is no evidence that the maintenance services are subject to any verification and validation), and there are no records of this work. The debugging process is wrongly considered to be a testing process [Pressman 1997, s17.7.1].

## D.4 Level 2

**Foreword.** At level 2, the testing activities are distinct from the debugging activities. The testing activities are defined in the maintenance methodology and follow any change to the software. The tests are planned, but, due to the low maturity level, this planning is often carried out late or at the last moment. There is still a belief that testing is only required for source code [Burnstein et al. 1996b; Staab 2002].

Evo4.2.1 The verification and validation activities are documented, known, and planned. The plans are documented according to a documented procedure [S3<sup>m®</sup>; ISO 90003, 5.7.2; ITIL 2007c, 4.5.5].

At this maturity level, the activities for testing are described and the responsibilities known. The definition of the tools and environments is known (see practice Req2.2.10). A process is set up and documented (see the sample in [Dias et al. 2003]). The template for the test plan is created at this maturity level. The techniques for testing and reviews are examined and documented. The planning of reviews and tests must be carried out in each project for transition and for requests for modification (see the Req2.3.20 practice). So it is expected that specific activities for verification and validation have been identified in the formal plans for testing a request. It is also expected that the organization knows the important aspects that influence the tests and ensure their success [Gilb 1995]. These plans are coordinated with other plans if needed (see the Req2.2.9 practice). These simple goals are established, and the responsibility for each type of test is assigned. The complexity of the plans change according to the effort associated with a change.

### **Reviews Road Map**

Evo4.2.2 Reviews and verification/validation are performed, selectively, after the production of some key intermediate products [S3<sup>m®</sup>; ISO 90003, 7.3.1 b].

At this maturity level, the maintenance organization performs selective peer reviews on some products judged to be more critical. The review is not necessarily performed independently, but it is performed formally and records are accessible on request. Formal reviews and detailed designs (impact analysis, in this case) will result in better quality.

Evo4.2.3 Supplier maintenance activities are selectively reviewed at given steps [SEI 2002, PA166.IG102.SP102.SubP103.N101].

Reviews can be formal or simple, and include the following steps:

- Prepare for the review
- Ensure that the pertinent stakeholders attend the review
- Manage the review
- Identify, document, and track all the items to be corrected after the review
- Prepare and distribute the minutes of the review to pertinent stakeholders

The SEI identifies the technical review that is aimed at verifying whether or not the intermediate product meets the standards and requirements. [SEI 2002, PA166.IG102.SP102.SubP104]. This technical review includes the following activities [SEI 2002, PA166.IG102.SP102.SubP104.N101]:

- Ensuring that the subcontractor has understood well the needs and requirements of customers and end users
- Reexamination of the technical activities, verification that the subcontractor has interpreted and implemented the detailed designs well, and determination that the products conform to standards
- Ensuring that the technical responsibilities are met and that the technical questions are communicated and solved in an opportune way
- Supplying appropriate information and technical support to the supplier

The SEI also identifies a review for management that can be used to reexamine the agreements with the subcontractors, taking into account the contracts [SEI 2002, PA166.IG102.SP102.SubP105]. This review for management typically includes the following activities [SEI 2002, PA166.IG102.SP102.SubP105.N101]:

- Reexamination of the critical dependencies
- Reviewing the risks associated with the work

The SEI recommends that planning for these reviews be carried out jointly [SEI 2002, PA166.IG102.SP102.SubP105.N102]. The maintainer uses the review results to improve the execution of the supplier and establish long-term relationships with some preferred suppliers [SEI 2002, PA166.IG102.SP102.SubP106].

### ***Acceptance Tests Road Map***

Evo4.2.4 The acceptance test is performed according to a documented procedure to show that the modified product conforms to and meets the customer's requirements [ISO 90003, 5.7.4 and 5.7.5].

The acceptance test (including the functional tests, the performance tests, and the acceptance and installation tests) for the regular requests is performed and documented to show that the software modification meets the customer's requirements.

Evo4.2.5 Regression tests are systematically and formally performed on all parts of the software affected by modifications, including corrections [ISO 14764, 8.3.2.2; S3<sup>m®</sup>].

Bennett [2000] indicates that there are other activities unique to maintenance: studies of various types of requests for change supported by a help desk call center and its support software, the activities for impact evaluation of a change, and specialization in tests and regression verification.

ISO 14764 identifies a specific need during maintenance, which is to ensure that the existing components of the software that have not been subject to change are verified to ensure that they have not been indirectly affected by those changes [ISO 14764, 8.3.2.2 a and b]. The maintainer will use techniques here for test regression to verify the whole system.

### ***Installation (Move to Production of a Change or Version) Road Map***

Evo4.2.6 The coordination of the installation of the test on-site and of the software maintenance with the user are carried out according to a documented procedure [ISO 90003, 5.9.3; S3<sup>m®</sup>].

The final implementation is performed with the agreement of the customers, and follows a documented procedure.

Evo4.2.7 The practices surrounding the installation of software modifications (including the installation documentation) are designed to reduce interruptions of the daily work of the system [S3<sup>m®</sup>].

The implementation must ensure continuous service.

Evo4.2.8 The maintenance engineer participates in acceptance tests during the final transition of software from the subcontractor and supplier, according to a documented procedure [ISO 90003, 6.7.3; S3<sup>m®</sup>].

This is the final opportunity to participate in the activities for the final deployment of the application software.

# Chapter 11

## Exemplary Practices— Support to Evolution Domain

### This chapter covers:

Process Domain	Key Process Area	Road Maps
Support to Evolution Engineering	Configuration and version management	<ul style="list-style-type: none"><li>• Change management</li><li>• Baseline configuration</li><li>• Reservation, follow-up, and control of products</li></ul>
	Process, service, and software quality assurance	<ul style="list-style-type: none"><li>• Evaluate objectively</li><li>• Identify and document nonconformances</li><li>• Communicate nonconformances</li><li>• Follow up on corrections/adjustments</li></ul>
	Maintenance measurement and analysis	<ul style="list-style-type: none"><li>• Define measurement program</li><li>• Collect and analyze measurement data</li><li>• Create repository of maintenance measures</li><li>• Communicate measurement analysis</li></ul>
	Causal analysis and problem resolution	<ul style="list-style-type: none"><li>• Investigate defects and failures</li><li>• Identify and analyze causes</li><li>• Propose solutions</li></ul>
	Software rejuvenation, migration, and retirement	<ul style="list-style-type: none"><li>• Redocument</li><li>• Restructure</li><li>• Reverse engineer</li><li>• Reengineer</li><li>• Migrate</li><li>• Retire</li></ul>

## 11.1 CONFIGURATION AND CHANGE MANAGEMENT—DETAILED EXEMPLARY PRACTICES

**Foreword.** The maintainer, because of the nature of his work, does not have to develop a configuration management plan for each request. Typically, the software is delivered with its configuration management. In the case in which the configuration management is missing, it will need to be developed and implemented. It is in the interests of both the maintainer and the developer, working for the same IS/IT organization, to use the same configuration management tools and procedures. It was stated earlier (refer to the predelivery and transition KPA) that it is in the maintainer's strategic interest to become involved early in understanding and using the configuration management process and tools. A software product that is placed under configuration management includes products delivered to the customers, documents and intermediate products from the maintainer's life cycle, products acquired from a supplier, tools, and other items used to create and describe these products [SEI 2002, PA159.N102].

### E.1 Level 0

Sup1.0.1 The software maintenance organization does not perform software configuration management activities [S3<sup>m®</sup>].

The maintenance organization does not include a configuration management process as part of its methodology. IS/IT and maintenance management do not recognize the value-added component of this process.

### E.1 Level 1

**Foreword.** Level 1 is characterized by a chaotic process, executed on standard platform libraries and files. Unsupervised employees perform configurations in a directory structure. There is no interference from qualified personnel and no use of specialized tools.

Sup1.1.1 The software maintenance organizational unit does not perform software configuration management activities in a structured and coordinated manner [S3<sup>m®</sup>].

Each software maintenance engineer is taking care of the software artifact's location. Typically, the artifact and source code are located in folders using a personal and undocumented structure.

## **E.1 Level 2**

**Foreword.** At level 2, configuration management software to which only the maintenance employees have access is being used. Procedures vary by application system. There is a follow-up of the change identification and control process, mainly centered on MRs and the source code. Often, many different configuration management tools are used, depending on the platform or for historical reasons. Configurations are constructed with these tools by unsupervised employees.

### ***Change Management Road Map***

Sup1.2.1 Modification requests are documented, prioritized, and authorized by the customers before implementing a change to the application software [S3<sup>m®</sup>].

At this maturity level, modifications to applications must be authorized before they are implemented. Changes are treated individually, and there is a meeting (or contact) to inform the customer (see also Req2.2.15 and Req2.2.18).

### ***Baseline Configuration Road Map***

Sup1.2.2 The configuration management plan, documented and approved by the developer (if available), is used as a basis for starting the maintainer's configuration management activities [S3<sup>m®</sup>].

In order to establish an application's basic configuration, the developer's plan (see practice Req2.2.9) and the tools proposed (see practice Req2.2.10) are assessed and must be considered first. This configuration must also be obtained before the application's final transition (see practice Evo1.2.10).

Sup1.2.3 Configuration management systems are used to handle production versions of the software maintained [ITIL 2007c, 4.3; Niessink et al. 2005, 5.5 activities 3 and 4].

The maintainer uses many automated support applications to manage the configuration of its application software. These ensure secure source code storage and safe, controlled extraction of current and prior versions [Camélia 1994, 6.7.2.4 and 5]. They also keep records on the state and history of modifications for all elements and configuration images [Camélia 1994, 6.7.2.6].

Sup1.2.4 The maintained source code is handled by configuration management control [S3<sup>m®</sup>].

The maintainer's configuration management formally handles the software's source code.

### ***Reservation, Follow-up, and Control Road Map***

Sup1.2.5 Documentation management is in place in the maintenance organization for each MR (creation, modification, version, archiving, diffusion, destruction, and validation) [S3<sup>m®</sup>].

Sup1.2.6 Configuration elements and the configuration image are modified formally, according to a documented procedure [ISO 90003, 6.1; S3<sup>m®</sup>].

The configuration software's procedure is followed.

Sup1.2.7 There is traceability between the CRs, the requirements, the detailed designs, and the source code at one end, and the integration testing sets at the other [S3<sup>m®</sup>].

Sup1.2.8 Standardized reports that document configuration management activities and the content of the images produced are developed and made available to the concerned groups and people [SEI 2002, PA159.IG103.SP101; Niessink et al. 2005, 5.5 activity 10].

## **11.2 PROCESS, SERVICE, AND SOFTWARE QUALITY ASSURANCE—DETAILED EXEMPLARY PRACTICES**

**Foreword.** Quality assurance is often absent in maintenance organizations [Pigoski 1997, p. 173]. Its main interest is the conformity of internal processes used in everyday work. Quality assurance analysts must minimize the impact of reviews in order to avoid affecting/slowing down operational work [SEI 2002, PA145.N103]. To ensure valid conclusions, measures, follow-ups, and control of activities and general improvement, objectivity in the quality assurance process and in evaluation are critical. This objectivity is achieved through independence and the use of criteria. Depending on their size, maintenance organizational units will or will not have an internal quality assurance function [SEI 2002, PA145.N104]. If the quality assurance function is part of the organization's maintenance unit, many issues must be addressed in order to ensure its objectivity. All personnel resources performing quality assurance activities must be competent and trained in that subject. This function must be separated from those directly involved in daily software maintenance (however, it is possible to use the resources that support application A to per-

form quality assurance on the work being done on application B). In all cases, there must be a structure that allows the reporting of nonconformities to the management team [SEI 2002, PA145.N105].

## **E.2 Level 0**

Sup2.0.1 Quality assurance is not performed by the software maintenance organization [S3<sup>m®</sup>].

The maintenance organization does not have an independent quality assurance process. Management has not recognized the need for this process [IT Governance Institute 2007, M3].

## **E.2 Level 1**

Sup2.1.1 The maintenance organization performs quality assurance activities informally [S3<sup>m®</sup>].

The maintenance organization assesses its processes/services and makes changes to software locally and independently from the other software maintenance units. Maintenance resources are in charge of their work and it is only when a problem arises (often brought up by an external actor or a customer) that the maintainer performs a review of the problem's cause or of a specific nonconformity. Unqualified personnel perform quality assurance interventions. This informal approach does not allow for nonconformities to be documented and be available for independent external review. [IT Governance Institute 2007, M3]

## **E.2 Level 2**

**Foreword.** At level 2, quality assurance activities are repeatable, but still intuitive. IS/IT has deployed quality assurance processes. Quality assurance mainly concerns development projects and very few maintenance activities. Independent reviews still occur reactively when there are software failures and/or user complaints. Quality assurance requirements depend on the organization's needs, and it is the IS/IT managers who establish priorities. Upper management supports this activity.

Sup2.2.1 Responsibility, authority, and the relationships between individuals in charge of managing, executing, and verifying the work who have an influence on quality, are defined [ISO 9001, 4.1.2.1; SEI 2002, GP106; S3<sup>m®</sup>].

Sup2.2.2 A quality assurance plan that includes the software maintenance processes/services is established, documented, approved, and monitored [SEI 2002, GP110; ISO 90003, 4.2.3 and 5.5; IEEE Std. 730 and 983; xxxx xxxx 2005, 6.1(a5) and 6.2(a5); Niessink 2005, 5.7 activity 1].

### **Objective Evaluation Road Map**

Sup2.2.3 Independent conformity reviews are performed on critical products or major changes [S3<sup>m®</sup>].

Sup2.2.4 Quality assurance activities are performed in accordance with the maintenance methodology criteria [SEI 2002, GP113; S3<sup>m®</sup>].

Sup2.2.5 The maintenance organization is audited to ensure their conformity with the defined and accepted process [SEI 2002, PA150.IG101.N102; S3<sup>m®</sup>].

Sup2.2.6 Product configuration management audits are performed in accordance with a documented procedure [SEI 2002, PA159.IG101.N101; S3<sup>m®</sup>].

### **Identify and Document Nonconformance Road Map**

Sup2.2.7 Products are maintained in accordance with a documented procedure, covered by the quality system [ISO 90003, 5.9 and 5.10; S3<sup>m®</sup>].

Sup2.2.8 The maintenance organization monitors the suppliers' and developers' quality assurance activities, in accordance with a documented procedure [SEI 2002, PA166.IG102.SP102.SubP101; S3<sup>m®</sup> 10].

The supplier's quality assurance function regularly reviews the supplier's activities and results with the client's quality assurance personnel [S3<sup>m®</sup>].

Sup2.2.9 Samples of intermediate and final maintenance products are reviewed for their conformity to customer request requirements [SEI 2002, PA150.IG103; S3<sup>m®</sup>]

Sup2.2.10 Nonconformances observed in maintenance activities are documented and handled locally, in accordance with a documented procedure [SEI 2002, PA145.N107; ISO 9001, 4.4; S3<sup>m®</sup>].

***Communicate Nonconformance Road Map***

Sup2.2.11 Results of quality assurance activities are the object of periodical reports presented to the maintenance management and engineers [SEI 2002, PA145.IG102.SP101; Niessink et al. 2005, 5.7 activity 6].

Sup2.2.12 Independent conformity review results are periodically reviewed with upper management [S3<sup>m</sup>®].

***Follow-up on Corrections and Adjustments Road Map***

Sup2.2.13 The maintenance engineers who manage, execute, and verify work that has an impact on quality has the necessary liberty and authority to report quality issues [ISO 9001, 4.1.2.1; SEI 2002, PA145.EL115].

Maintenance engineers have the liberty and authority to:

- Take measurements in order to prevent product nonconformity
- Identify and record any quality problems pertaining to the product
- Elicit, recommend, and supply solutions by using previously established communication channels
- Verify the implementation of solutions
- Manage the handling, delivery, and installation of a nonconforming product until its faults or unsatisfactory elements have been corrected

**11.3 MAINTENANCE MEASUREMENT AND ANALYSIS—DETAILED EXEMPLARY PRACTICES**

**Foreword.** The S3<sup>m</sup>® model is about common small maintenance activities, not larger-scale activities that should be managed as projects, using proven project management techniques. Project management is only necessary in large maintenance projects and not for maintenance requests. For specific cases of large maintenance projects, the CMMi model should be used. Software rejuvenation activities can possibly become too big to be managed by the maintenance team. Therefore, this road map should be used carefully and by taking the particular situation into account.

**E.3 Level 0**

Sup3.0.1 The maintenance organization does not measure process/services, software, or personnel [S3<sup>m</sup>®].

The maintenance organization does not analyze or measure its activities or resources. There are no estimates, objectives, or effort tracking. The need to understand the process objectives is not yet recognized [IT Governance Institute 2007, MEI].

### E.3 Level 1

Sup3.1.1 Measurement and analysis, in the maintenance organization, is performed individually by each manager [S3<sup>m®</sup>].

Maintenance managers recognize the need to collect and analyze maintenance metrics. Standardized processes for collection and analysis have not yet been identified. Analyses and measurements are chosen and performed on a case-by-case basis, according to specific customer or management needs. These analyses are mainly performed after a software failure or another problem that has put the maintenance organization on the defensive. Measurement and analysis are present in other IS/IT organizations, but not yet in software maintenance. Measures of software, resources, and processes/services specific to software maintenance have not yet been developed.

### E.3 Level 2

**Foreword.** At level 2, basic measurements and analyses have been defined and implemented. Collection and analysis procedures are defined locally and vary from one maintenance organization to another. Analyses are still reactive and depend on specific individuals. Accesses to data collection tools are limited and/or not yet used to their full capacity. The maintenance organization performs limited analyses with a single-minded vision and without analyzing its real contribution in terms of costs and benefits, such as its contribution to revenue generation.

Sup3.2.1 The maintainer has defined basic operational measurements that meet software maintenance measurement objectives [SEI 2002, PA154.IG101.SP102; ISO 90003, 8.2]

To meet this practice, the chosen basic measurements must meet measurement objectives. See the definition of basic measurements in practice Pro4.2.1 also. At this maturity level, measurement is simple and mainly comprises:

- Surveys (see Pro1.2.3)
- Qualitative observations (see Pro1.2.4)
- Internal benchmarks (see Pro1.2.5)

- Failure data (see Pro1.2.6)
- ISO 9001 audits (see Pro1.2.7)
- Process evaluation (see Pro1.2.8)

Sup3.2.2 The maintainer performs analyzes and produces reports to address specific requests [S3<sup>m®</sup>; ISO 90003, 7.6].

At this maturity level, simple analyses and reports are developed and documented. Employees have been trained in basic analysis (see practice Pro1.4.2). There is not much contextual information, however, and its use is neither generalized nor frequent. Measurements are generally used internally and analyses are performed on demand, mostly when trying to explain service failures or problems. Analysis diffusion is often restricted and its interpretation limited to the actors involved.

## **11.4 CAUSAL ANALYSIS AND PROBLEM RESOLUTION—DETAILED EXEMPLARY PRACTICES**

**Foreword.** The S3<sup>m®</sup> model is about small software maintenance activities, not larger-scale activities that should be managed as projects, using proven project management techniques. Project management is only necessary in larger maintenance projects and not for small maintenance requests. For the specific case of large maintenance projects, the CMMi model should be used. For example, software rejuvenation activities can possibly become too big to be managed by the maintenance team. Therefore, this road map should be used carefully and by taking the particular situation into account.

### **E.4 Level 0**

Sup4.0.1 The maintenance organization does not recognize the need for a problem resolution and causal analysis process [S3<sup>m®</sup>].

Maintenance management does not recognize the need for a problem resolution and causal analysis process. Resolution of problems is left to each individual to handle in a reactive way [IT Governance Institute 2007, DS10].

### **E.4 Level 1**

Sup4.1.1 The maintenance organization recognizes the need to have a problem resolution and causal analysis process [S3<sup>m®</sup>; ISO 90003, 8.3].

The maintenance organization has recognized the need to solve problems permanently and to evaluate their cause(s). They also recognize that problem resolution is a process shared among many customer support organizations. Consequently, maintenance engineers are assigned, according to their domain of expertise and responsibilities, in order to work on solving problems and identifying their causes. The information is not yet shared with other organizations and solutions vary from one maintenance engineer to another, which creates other problems and results in a loss of productivity. Maintenance managers frequently change the focus and direction of their personnel regarding the next problem to fix [IT Governance Institute 2007, DS10].

## E.4 Level 2

**Foreword.** At level 2, IS/IT customer support organizations as well as maintainers are more conscious of the need to manage problems and eliminate their causes. The problem resolution process has evolved and is documented locally. Some maintainers are responsible for managing problems, finding their causes, and performing decisional analyzes. Information is shared among personnel. However, the IS/IT global problem management process is not structured—it is simple and reactive. The level of service to the user community varies and is hindered by insufficient knowledge and organizational barriers between the different IS/IT customer support organizations. Problem management reports, causes, and decisional analysis are simple and limited. [IT Governance Institute 2007, DS10]

### ***Investigate Defects and Failures Road Map***

Sup4.2.1 The software maintenance organization documents its problem resolution and causal analysis process [Kajko-Mattsson 2001, ME-Process-PCI-1 and ME-Process-RCA-1].

At this maturity level, there is more concise documentation that describes the resolution process and causal analysis process, and interfaces with other organizational units in IS/IT support. This process may still vary between maintenance organizations and software.

### ***Identify and Analyze Causes Road Map***

Sup4.2.2 Maintenance management analyzes quality, customer, and performance data regularly [S3<sup>m®</sup>; Malcolm Baldrige 2005, 2.1, 2.2 and 3.2(a3); ISO 90003, 8.4].

To meet this practice, decisional analysis activities on general quality, performance, and satisfaction data are identified. Results are communicated and can be the object of specific problem resolution assignments.

Sup4.2.3 Upper managers analyze the results of surveys and develop and implement an action plan based on their analysis. These results and the action plan are communicated to all personnel [S3<sup>m®</sup>; Malcolm Baldridge 2005, 1.1(b1)].

To meet this practice, decisional analysis activities on general maintenance personnel satisfaction data are identified. Results are communicated and can be the object of specific problem resolution assignments.

Sup4.2.4 Process analysis and reports are produced and distributed to the concerned groups [S3<sup>m®</sup>].

## **11.5 SOFTWARE REJUVENATION, MIGRATION, AND RETIREMENT—DETAILED EXEMPLARY PRACTICES**

**Foreword.** The S3<sup>m®</sup> model is about common maintenance activities, not larger-scale activities that should be managed as large projects, using proven project management techniques. Project management is only necessary in large maintenance projects and not for maintenance requests. Software maintenance projects are subject to the CMMi model recommendations. Software rejuvenation activities can possibly become too big to be managed by the maintenance team. Therefore, this road map should be used carefully and by taking the particular situation into account.

### **E.5 Level 0**

Sup5.0.1 The software maintenance organization does not perform rejuvenation activities on the maintained software [S3<sup>m®</sup>].

### **E.5 Level 1**

Sup5.1.1 In the software maintenance organization, rejuvenation is performed informally, during a specific change, without any prior planning [S3<sup>m®</sup>].

Application software must be periodically maintained. Redocumentation, restructuring, and retro engineering or reengineering decisions are made by the resource making the change. The software works, and employees recognize the fact that the documentation is outdated [Pressman 2005]. These activities are limited and do not have a significant impact on maintainability or on future software maintenance costs. The maintainer takes charge of this activity on his own initiative, and it is only rarely recognized by customers.

Sup5.1.2 Retirement of a feature or application is carried out by leaving all software component infrastructures in place and preventing its execution [S3<sup>m®</sup>].

At this maturity level, software that is no longer used is only commented on. This practice is controversial because of the unused source code's clutter, though it still is a common, low maturity practice.

## E.5 Level 2

### ***Redocumentation of Product Road Map***

Sup5.2.1 Maintained product redocumentation is only performed for components subjected to a change or correction [S3<sup>m®</sup>].

This redocumentation practice is based on the use of an opportunity for change or correction [Pressman 2005]. The maintenance analyst will use that opportunity to improve the software documentation with the information he or she possesses for a specific request. This technique must be used from the start because the application will go through many changes in its first years of operation. Adopting this approach at the end of the software life cycle will yield no valid results. This approach is based on minimal effort, and it will be considered that, over the years, redocumentation has been performed with little effort.

### ***Restructuring the Product Road Map***

Sup5.2.2 Maintained product restructuring is only performed for components subjected to a change or correction [S3<sup>m®</sup>].

This restructuring practice is based on the use of an opportunity for change or correction. The maintainer will use that opportunity to restructure parts of the software (source code and data structures) that are affected by a specific request. The work is done manually and locally. This approach is based on minimal effort and on the same principle as practice Sup4.2.1.

### ***Reverse Engineering the Product Road Map***

Sup5.2.3 Maintained product reverse engineering is only performed for components subjected to a change, a correction, or an operational support request [S3<sup>m®</sup>].

This reverse engineering practice is based on the use of an opportunity for change or correction. The maintainer will use that opportunity to create new documentation

for the parts of the software that are affected by a specific request (e.g., specifications, design, data models). This information is formatted and inserted into the software's official documentation. This approach is based on minimal effort and on the same principle as practice Sup4.2.1.

At this maturity level, maintainers will make diagrams describing the software's behavior to help represent software execution during a specific change or correction request. This retro-engineering effort is not necessarily supported by a tool. The activity is also present when the maintainer tries to extract an application's business rules, to help a customer better understand the way in which it functions.

### ***Reengineering the Product Road Map***

Sup5.2.4 Maintained product reengineering is only performed for components subjected to a change, a correction, or an operational support request [S3<sup>m®</sup>].

This reengineering practice is based on the use of an opportunity for change or correction. The maintainer will use that opportunity to create new documentation or an improved source code design without modifying software functionality. This effort is limited to parts of the software affected by a specific request (or its minimal set of components).

### ***Software Migration Road Map***

Sup5.2.5 The maintenance organization develops a migration plan and communicates it to the customers for approval [ISO 12207, s5.5.5; ISO 14764, s8.5.2.2].

The maintainer will have to develop a migration plan that will be authorized and used for this project. The plan will contain, for example [ISO 12207, 5.5.5.2]:

- Requirement studies
- Development of migration tools
- Conversion of products and data
- Migration execution
- Migration verification
- The old software's support terms

The maintainer must communicate the plan and the activities to his customers. These communications must include, for example [ISO 12207, 5.5.5.3]:

- A statement explaining why the old environment is no longer supported
- A description of the new environment and its availability date

- A description of support services that will be available when the old environment is no longer available

The maintainer must consider the use of both systems in parallel, in order to obtain a better transition [ISO 12207, 5.5.5.4]. Training is given during this period. There must be proper communication with the customers, and, when the new environment is implemented, products (source code and documents) must be archived [ISO 12207, 5.5.5.5]. The maintainer will perform the post mortem revisions and ensure that the old environment's data are accessible, secure, and protected in the new environment [ISO 12207, 5.5.5.6, 5.5.5.7].

Sup5.2.6 Software tools are used to support migration from one technological platform to another [S3<sup>m®</sup>].

Code translators are used [Camélia 1994, 7.1.2.8]. Migration of configuration elements of a product supported by one software engineering tool to another such tool is executed in a partially automatic manner [Camélia 1994, 6.10.2.2].

### ***Software Retirement Road Map***

Sup5.2.7 The maintenance organization develops a software retirement plan and communicates it to the customers for approval [ISO 12207, s5.5.6; ISO 14764, s8.6].

The maintainer will have to develop a software retirement plan that will be authorized and used for this project. The plan will address, for example [ISO 12207, 5.5.6.1]:

- The end of support (complete and partial) at a given date
- Archiving of software products and documentation
- Responsibilities for activities following software retirement
- Transition to a new application, if necessary
- Accessibility of archived data

The maintainer must communicate the plan and the activities to the customers. Communications must include, for example [ISO 12207, 5.5.6.2]:

- A description of the replacement software and its availability date
- An explanation of why the software is no longer supported
- A description of support alternatives available after the application's retirement

The maintainer must consider the use of both systems in parallel, in order to obtain a better transition [ISO 12207, 5.5.6.3]. Training is given during this period. There must be proper communication with the customers, and, when the new environment is implemented, products (source code and documents) must be archived [ISO 12207, 5.5.6.4]. The maintainer will perform the post mortem revisions and ensure that the old environment's data are accessible, secure, and protected in the new environment [ISO 12207, 5.5.6.5].

# Chapter 12

---

## Assessment Process, Assessment Tool, and Case Studies of the Use of S3<sup>m®</sup>

### This chapter covers:

- S3<sup>m®</sup> maturity evaluation process overview
- S3<sup>m®</sup> small-setting assessment tool
- Four case studies

In this chapter, the four verification and validation (V&V) steps of the S3<sup>m®</sup> process model are presented. An evaluation, leading to four case studies in which the software maintenance maturity model was used in a Middle East telecommunications company, will be described.

### 12.1 EVALUATION PROCESS AND SUPPORT TOOLS

Since 1999, software engineering maturity models have been governed by the ISO 15504 reference model. To claim an ISO-conformant software process evaluation, the model's conformity with the ISO 15504 reference model must be verified. The five process categories proposed by ISO 15504 are grouped according to the software life-cycle process dimensions defined in ISO 12207: the customer–supplier and engineering processes are part of the primary ISO 12207 life cycle, the support process is part of the ISO 12207 support life cycle, and the two management and organization processes are part of the ISO 12207 organizational life cycle. S3<sup>m®</sup> also takes the same perspective as that presented in Figure 1.13.

ISO 15504 also standardizes six descriptive process components for maturity models:

1. *An Identifier*, which identifies one of the five categories, followed by a unique sequential number structured from the general to the specific (e.g., CUS 1: Acquisition process, CUS 1.2: Supplier selection process)
2. *A Name*, which is a short sentence encapsulating the process objective
3. *A Process type* (from 1 to 5):
  - 1—Base process, which is identical to the ISO 12207 process
  - 2—Extended process, which adds to ISO 12207's existing processes
  - 3—New process, which is not covered by ISO 12207
  - 4—Component, which is an activity (or a group of activities) from an existing ISO 12207 process
  - 5—Component extension, which adds to the activities (or group of activities) of an existing ISO 12207 process with additional elements
4. *Process goal*, which is a descriptive text about the high-level goals of the process
5. *Expected process results*, to describe the observable results of a correct process implementation. These must refer to a list of items which in turn must appear immediately after “following a success in process implementation”
6. *Note*, containing additional information about the process

Capability level implementation is performed with the help of a set of attributes. These attributes are used to determine whether or not a process has reached a certain capability level. Each attribute measures a specific aspect of process capacity on a percentage scale. The standardized scale for evaluated attributes consists of the following irregular intervals:

- N: Not reached: 0–15%
- P: Partially reached: 15%–50%
- L: Mostly reached: 51%–85%
- F: Entirely reached: 85%–100%

ISO 15504 defines the required set of conditions for conducting planning and executing an evaluation. If an evaluation is conducted in such a way that the ISO 15504 conditions are met, then the evaluation is said to be ISO 15504 conformant.

The ISO 15504 standard specifies the set of required conditions for repeatable, reliable, and coherent evaluation results. This standard's goal is to increase maturity measurement repeatability. In an evaluation context, repeatability is defined as getting the same maturity level whenever the evaluation procedure is repeated.

An evaluation support prototype named S<sup>3mAssess®</sup> was developed and used to support assessments. The prototype permitted selection of a KPA, use of the model practices directly, identification of the level of conformance, and insertion of comments to describe the reasoning behind the rating.

The S3<sup>m®</sup> assessment plan and the use of an evaluation support tool ensured that the following obligations of ISO 15504 were fulfilled:

Phase de réponse: choisir les réponses appropriées

D1\_K1 | D1\_K2 | D1\_K3 | D1\_K4 | D1\_K5 | Analyse | 1

**2** Niveau 0

1.1.0.1	<input type="radio"/> Oui <input type="radio"/> Non
---------	--

Niveau 1

1.1.1.1	<input type="radio"/> Oui <input type="radio"/> Non
1.1.1.2	<input type="radio"/> Oui <input type="radio"/> Non

Niveau 2

11.2.1	ON	OP	OL	OF
11.2.2	ON	OP	OL	OF
11.2.3	ON	OP	OL	OF
11.2.4	ON	OP	OL	OF
11.2.5	ON	OP	OL	OF
11.2.6	ON	OP	OL	OF
11.2.7	ON	OP	OL	OF

Niveau 3

11.3.1	ON	OP	OL	OF
11.3.2	ON	OP	OL	OF
11.3.3	ON	OP	OL	OF
11.3.4	ON	OP	OL	OF
11.3.5	ON	OP	OL	OF
11.3.6	ON	OP	OL	OF
11.3.7	ON	OP	OL	OF
11.3.8	ON	OP	OL	OF
11.3.9	ON	OP	OL	OF

Niveau 4

11.4.1	ON	OP	OL	OF
11.4.2	ON	OP	OL	OF
11.4.3	ON	OP	OL	OF
11.4.4	ON	OP	OL	OF
11.4.5	ON	OP	OL	OF
11.4.6	ON	OP	OL	OF
11.4.7	ON	OP	OL	OF

**3** Pratiques

11.0.1

Descriptions

EST-CE QUE LA DECLARATION SUIVANTE DECRIIT VOTRE SITUATION DE TRAVAIL. L'UNITE ORGANISATIONNELLE DE MAINTENANCE DU LOGICIEL NE FAIT PAS D'ACTIVITE D'AMELIORATIONS STRUCTUREES DES PROCESSUS MENANT A DES AMELIORATIONS DE PROCESSUS PERISTANTES ET CONTRROLEES. L'organisation n'a pas de processus de planification et d'amélioration de la qualité et une méthodologie de cycle de vie de la maintenance du logiciel. La direction des technologies de l'information ainsi que les dirigeants de la maintenance du logiciel ne reconnaissent pas qu'un

**4** Commentaire de l'évaluateur

OK | Abbrechen | Übernehmen | Hilfe

**Figure 12.1** S3<sup>Assess</sup> evaluation support tool.

- Evaluation inputs are documented and include at least:
  - Identity of the evaluation's sponsor
  - The maximum capability level investigated for each process
  - Organizational units that use these processes
  - Contextual data (size, demographics, scope of application, criticality and complexity of delivered products, and product quality)
  - Evaluation limits (in terms of resource availability, maximum evaluation duration, processes excluded, sample size, result properties, and distribution restrictions)
- A description of the evaluation model being used
- The evaluators' identities and the chief evaluator's credentials
- The staff being evaluated and their responsibilities
- The process used for evaluation, which includes at least:
  - Planning
  - Data collection
  - Data validation
  - Maturity measurement for each process
  - Result report

6. Evaluation results to be kept, in particular:

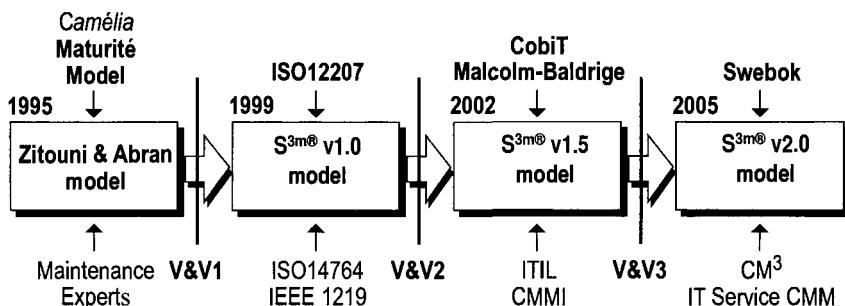
- Evaluation date
- Inputs
- Interview and review results
- Maturity measurements for each process
- The evaluation approach that was used

## 12.2 EXAMPLE OF EVALUATION RESULTS

Activities for evaluating the maturity of an organization using the S3<sup>m®</sup> maturity model have been taking place progressively since 1995 (see Figure 12.2). Initially, the software engineering research laboratory of the University of Québec in Montréal, sponsored by research grants from major Canadian telecommunications companies, developed an initial maturity model specific to software maintenance, which was published by Zitouni and Abran in 1995 [Zitouni 1995]. It was initially a theoretical model and was used internally in a large Canadian telecommunications company for assessing the maturity of software maintenance organizations with the objective of improvement. This was the first validation of the model, referred to as V&V1 in Figure 12.2.

During 1998, requests were received from other telephone/telecommunications companies that showed renewed interest in the software maintenance maturity model. A project was initiated to use the Zitouni and Abran model and include software maintenance standards. Iterative mapping activities led to an updated version (version 1.0), which took into account international standards on software maintenance (ISO 12207, ISO 14764, and IEEE 1219). Version 1.0 was published internally for the participating organizations during 1999. Experiments were next performed in an industrial setting under the sponsorship of a Middle East telecommunications company, referred to as V&V2 in Figure 12.2.

An evaluation process was prepared to ensure that a rigorous approach was used to establish a maturity level. This required additional resources to develop an evalua-



**Figure 12.2** Verification and validation of the S3<sup>m®</sup> model.

ation tool that would meet the ISO 15504 criteria listed in the previous section. These types of evaluations have the objective of building organizational consensus by proposing improvements and pointing out the weaknesses as well as the strengths that can be leveraged.

A senior manager (the IS/IT Planning Director) promoted the activity, and the team had already performed maturity evaluations and met the prerequisites for the composition of the team and the assignment of a trained team leader.

Evaluations were conducted by a team composed of one lead assessor, the software process improvement manager, the IS/IT planning senior manager, one software maintenance manager, one maintenance programmer, and one independent testing analyst.

Preparation took 4 weeks. An evaluation plan was developed that addressed the front-office and back-office areas of the company. The plan identified the scope, objectives, and participants. Evaluation objectives were determined by the IS/IT senior planning manager, and were limited to assessing the organization for level 1 and 2 practices. The plan was presented to the general manager and approved. The general manager then proceeded to call a meeting that included all the software maintenance personnel to explain the activity and discuss the objectives of the evaluation. A discussion item raised at this meeting was that one of the objectives of the evaluations was to identify shortcomings in the S3<sup>m®</sup> version 1.0 maturity model, particularly regarding coverage, usability, and applicability, and that personnel inputs were welcome.

Process evaluations were conducted over four consecutive days, during which time each of the four process domains of S3<sup>m®</sup> were rated.

This V&V2 activity took place just prior to 2002, at which time version 1.0 of the model was being used in four process evaluations of small maintenance groups (6 to 8 individuals). Each participant was also asked to fill out an observation/PR form where they could list positive and negative feedback about the model itself. The ISO 15504 trial comment ranking was used to help the participants formulate feedback by suggesting categories of issues. Six out of eight participants of the maintenance unit supplied observation/PRs. Their positive and negative (+/-) perceptions of the maturity model, as well as the evaluation and improvement activities, are summarized in Figure 12.3.

The organization was already involved in ISO 9001, Malcolm Baldrige, and COBIT audit activities, which gave rise to some alignment issues. The first feedback comment identified software maintenance practices from the quality and IS/IT audit perspectives, such as Malcolm Baldrige, the IT auditor guidelines (COBIT<sup>®</sup>), and the ITIL exemplary practices guidelines that were missing from version 1.0 of the model.

On the positive side, the participants felt that the S3<sup>m®</sup> version 1.0 model accurately identified their maturity and missing practices, and that it was useful in identifying what could be done next to improve the situation. Two participants also indicated that the model seemed, at that time, overweight for small maintenance units and that a shorter version would be more suitable. Software maintenance is often performed by a small team, and these maintainers expect small and simple models

Rank	Comments	+/-
1	There are important areas that the model does not address (ITIL, COBIT and Malcom Baldridge)	-
2	The model accurately portrayed the process state of the organization	+
3	The model was useful for identifying what has to be improved	+
4	Model has too many practices to be used in a small maintenance organization	-
5	Some KPAs have too many practices in order to achieve the level	-

**Figure 12.3** Perception of the usefulness of S3<sup>m®</sup>.

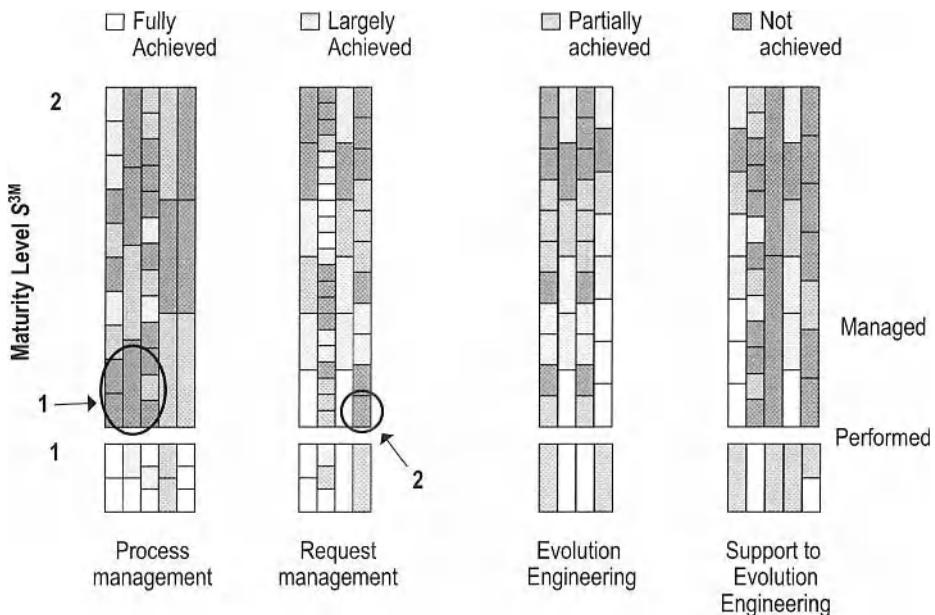
and tools. Finally, two participants felt that the PRO3 and REQ3 KPAs had a significantly larger number of practices (21) compared to other areas of the model. They referred to this problem as “unbalanced KPAs.”

The resulting maturity profile and evaluation report was presented to the IS/IT planning director, who discussed its findings with the maintenance personnel. The processes not meeting level 1 requirements were discussed as potential process improvement objectives.

After discussing the evaluation results with personnel and the general manager, the IS/IT planning director chose to initiate improvements in three areas. Two of the areas match the weaknesses identified in Figure 12.4: (1) software maintenance processes/services definition, and (2) SLAs. The third improvement activity that was decided upon was outside the scope of the evaluation. The organization urgently needed, during its current transformation, the capability of assessing the quality of the third-party software to be acquired by the organization.

These process improvement activities were initiated, and the maturity model guided the setting of objectives for each initiative. The software improvement initiatives have subsequently been published in the following papers:

- “Software Product Measurement for Supplier Evaluation,” presented at the Software Measurement Conference (FESMA–AEMES), Madrid, Spain, October 18–20, 2000 [April and Al-Shuroogi 2000].
- “Software Maintenance in an SLA: Controlling the Customer Expectations,” presented at the Fourth European Software Measurement Conference (FESMA2001), Heidelberg, Germany, May 8–11, 2001 [April 2001].
- “Software Maintenance Productivity Measurement: Controlling the Customer Expectations,” presented at IWSM 14th International Workshop on Software Measurement and Metrikon Congress, November 2–5, 2004, Berlin, Germany [April 2004c].
- “Assessment Results Using the Software Maintenance Maturity Model



**Figure 12.4** Case study S3<sup>m®</sup> maturity profiles for levels 1 and 2.

(S3<sup>m®</sup>)," presented at IWSM 16th International Workshop on Software Measurement and Metrikon Congress, November 2–5, 2006, Berlin, Germany [Paquette 2006].

The feedback and lessons learned from the first three case studies were used to produce S3<sup>m®</sup> version 1.5. The V&V3 step in Figure 12.2 was carried out in parallel with the 2004–2005 revision of the SWEBOk project, which included an update to the software maintenance body of knowledge. Additional mappings to external documents were inserted in this latest version of the S3<sup>m®</sup> model.

The S3<sup>m®</sup> content and validation results were published in papers presented at software engineering conferences in 2003–2004:

- Introduction of the classification of software maintainers' key processes [April 2004b]
- The model's purpose, scope, and high-level architecture [April 2004d]
- Introduction to the maintenance-related CMM proposals, the differences between this model and the CMMi, and the model creation process [April 2004b]
- Presentation of the maintenance context, the model's generic attributes, the detailed KPAs, and an example of one of the model's detailed practices for the Management of Service Requests and Events KPA [April 2004a]

## 12.3 FOUR CASE STUDIES USING S3<sup>m®</sup>

The following section describes the usefulness of the S3<sup>m®</sup> practices in each of the four process improvement projects of this experimentation. Further descriptions of those initiatives are provided in [April and Al-Shurogi 2000, April 2001, April 2004c, Paquette 2006].

### 12.3.1 Contributions to the Definition of Software Maintenance

The first process improvement initiative of the Middle East telecommunications company started on the basis that the evaluation had identified a weakness in the definition of software maintenance. The evaluation had revealed that users and customers did not know what software maintenance really did. This created frustration when maintainers did not agree to process requests. Figure 12.4 identifies this level 2 practice that has not been achieved (referred to as 1 in Figure 12.4). Management concurred that this aspect of the maintenance organization needed improvement to enhance customer/user relationships. The practice that has not been achieved is the following:

Pro2.2.1 There exists one description of maintenance processes/services that is used by the customer and by the software maintenance organizational unit of the organization.

The software maintenance organization has established and documented standardized processes for their internal activities.

Failing to perform this practice means that maintainers did not have a description that was used and understood by its customers. To help in describing maintenance processes/services, the team looked at more advanced practices to get a better idea of what this KPA was leading to.

The improvement team found useful information in the Pro2.3.2 practice that lists a number of maintenance processes/services and informs the reader about maintenance service categories documented in the international standards. With this guidance, the improvement team was able to define the services and align itself with those international standards. To learn more about this improvement initiative, refer to [April 2004c].

### 12.3.2 Contributions to the Definition of the Service Level Agreement (SLA)

The second process improvement initiative started on the basis that the evaluation had identified a weakness in service level agreements (SLAs). The evaluation had revealed that the maintenance organization did not have SLAs. This created confu-

sion as to where resources were assigned and what the priorities were when service was needed. The assessment had shown that this level 2 practice had not been achieved (referred to as 2 in Figure 12.4). Management concurred that this aspect of the maintenance organization needed improvement. The practice that had not been achieved is the following:

Req4.1.1 The agreements with clients, subcontractors, and outsourcers are missing or elaborated from unmodified templates, documents, and contracts from suppliers.

At this maturity level, awareness exists about the need to manage the SLAs and contracts, but the process is informal and reactive [IT Governance Institute 2007, ds1]. The responsibility for the control of the execution is informal. The measures are qualitative, with imprecise goals. The reports for performance are rare and inconsistent. The agreements have not been negotiated and are unilaterally implemented by the suppliers and subcontractors. The suppliers are in a strong position with respect to the agreements. This situation often results in an unequal relationship, characterized by blurry agreements, a lack of alignment in mutual agreements, few interviews or reports about the levels of services, nonexistent processes, and few foundations on which to build trust and to generate added value. The typical strategy for remedying the resulting problems is to change the supplier.

Failing to perform this practice means that maintainers did not have a description of the software or their relative priorities. Moreover, they did not have a coordinated interface with the help desk and computer operations. Software support services were not clearly defined and the responsibilities of the IS/IT organization and the users/customers were not clearly spelled out. To help produce an SLA, the team looked at more advanced practices in order to obtain a better idea of how this KPA was designed.

The improvement team found useful information in Req4.2.1, which explains that the introduction of more formal agreements must first be discussed with stakeholders. Req4.2.2 insists that a representative be assigned to the role of maintenance account manager to coordinate the establishment of SLA and customer/user communications. Req4.2.3 and Req4.2.5 state that a mutual agreement must be reached before initiating the work of building contracts or an SLA. Many level 3 practices describe the content that needs to be included in a software maintenance SLA. To learn more about this improvement initiative, refer to [April & Al-Shuroogi 2000].

### **12.3.3 Contributions to Software Product Quality Assessment During Predelivery and Transition**

The third improvement activity that the organization needed to perform urgently during its current transformation was outside the scope of capability evaluation. The evaluation had not revealed weaknesses in this area because this is an advanced

practice and was not assessed. The IS/IT planning director, however, needed a way to assess third-party software during predelivery and transition. The organization had chosen to acquire software, rather than develop it internally. They urgently needed quality assessment of the third-party software that was proposed to them.

The S3<sup>m®</sup> maturity model helped the organization once again. The practice that had not been achieved was the following level 3 practice:

Pro4.3.2 Products, intermediate products and application software are supported by quality measurement of their processes/services execution.

It is not necessary to measure all aspects of software maintenance; it might be too expensive and not necessarily efficient. Products and intermediate products like technical documentation, user documentation, tests, and application software must be measured. The maintainer must identify the product perspectives that have an impact on his customers and on the quality of application software. To achieve this practice, the maintainer must identify the product measures and explain why they must be collected and analyzed.

The intermediate products created by the maintainer, as well as the application software, must be measured. This refers specifically to the concepts of external and internal measures of software, as described in the quality model of ISO 9126. The external measures deal with the attributes of software quality that are perceived as important to customers; for instance, the SLA must be measured and reported on a regular basis. The organization must also measure and track the delivery times for requests and related functions. Measures of failure must be linked to the software modules and key processes that have failed. The data on products are collected and recorded in the process repository of the organization, according to a documented procedure. To achieve this exemplary practice, the whole set of organizational units for software maintenance must define the measures of software as specified in the services agreement and must ensure their validity.

The internal measures are more technical, and are important for documenting the internal software characteristics that have an impact on the external attributes as perceived by the customers; for example, even if the internal documentation of software is not seen by the customers, it has an impact on the time used by a resource to identify and locate a required change. It is, therefore, important to control internal attributes of software. To meet this exemplary practice, some basic internal attributes must be identified. How these internal attributes impact the external attributes and measures seen by the customers must be identified, as per ISO 9126. Each internal measure must be documented, and from where (e.g., from which activity or process) it is to be collected must be specified. The measurements will have to be collected at the operational level in daily activities. The required maintenance resources must be trained on software measurement; refer to Pro1.2.2.2. Data verification activities are also required to ensure that the data collected are valid.

Failure to perform this practice means that maintainers did not have the ability to define or measure the internal quality characteristics of the software in question. To

help in assessing the internal quality characteristics that have an impact on the maintainability of particular software, the team looked at advanced practices to get a better idea of how they could succeed in this task and what information sources could help them do so.

The improvement team found useful information in Pro4.3.8, which explains the performance models of software, and in Pro4.4.3, which discusses the instruments/tools used to achieve this goal. Many level 3 predelivery and transition practices describe the relationship that needs to be established with the supplier to ensure cooperation and smooth transition. To learn more about this improvement initiative, refer to [April 2000].

### **12.3.4 Contributions to the Improvement of a Very Small Maintenance Function**

In this other case study [Paquette 2006], the assessed organization offers IT services to a single large customer. A large portion of IT services support and maintain well-known software packages that have been adapted to, and implemented in, the customer's operations. Over the years, this organization has carried out less and less in-house development, and increasingly more support and maintenance. It has an ISO 9001:2000-certified quality system, which includes the IT services within its scope.

One of the software products the organization maintains is an invoicing process. This software application invoices the customers for their use of software, computer hardware, and telephone equipment. Recently, the software has experienced billing inaccuracies. The internal audit department asked for a verification of the accuracy of the invoicing, and requested a comparison of the results of a manual tally against the software system's invoice content. This audit highlighted a number of inaccuracies, mainly in the component inventory, including defects that sometimes generated losses for the organization, along with occasional customer overbilling.

A project was launched to address this issue. However, there was no plan in place for conducting it. Not only that, but a single staff member was deemed to be sufficient to conduct the impact analysis. This individual was the only person available at that time with experience in the programming language and database management system. Once the impact analysis had been completed, it was decided that additional functionality would be developed to address the data accuracy problem. To save time, and under management pressure to do this quickly, no documentation was produced, and maintenance and support, whenever necessary, were carried out by the single person assigned to the project. This new software has a number of users: the foremen, those in charge of the various inventories (software, hardware and telephone equipment), and one individual in the accounting department. The first version of the software helped monitor inventory changes and detect problems prior to billing. Errors found in this version were progressively corrected, and users identified various potential enhancements. The second version added functionality, within the same project constraints in terms of time pressure and staff. Fortunately, documentation was written for the system in this second version, including a user guide and a small PowerPoint presentation.

Maintenance remained the responsibility of the one person, progressively becoming a secondary task. This person left to fulfill other commitments, at which point the maintenance duties were transferred to another maintenance program. Unfortunately, knowledge transfer could not take place, with the result that the users returned to the old process.

This software is now an important part of the invoicing system; however, the users object to the inadequacy of the process, which is proving very difficult to carry out. Within this context of crisis management, the new maintainer can only successfully “extinguish fires,” which immediately reignite. As a result, management is beginning to recognize the importance of implementing sound maintenance practices.

#### **12.3.4.1 Assessment Procedure**

In some software maintenance organizations that are very small (sometimes consisting of a single maintainer), and in others for which a SCAMPI-type assessment was considered impractical, for example, in a setting in which maintainers want feedback on their processes but cannot devote three full-time weeks to an “official” CMMI-type assessment, an S3<sup>m®</sup> mini-assessment method has been developed. This method yields a reliable maturity rating without the investment of unavailable resources and also permits the selection of individual assessment components to focus the investigation on specific concerns and to tailor the scope of the assessment and rating effort to a level relevant to the individual software maintenance organization.

The S3<sup>m®</sup> model’s practices, for maturity levels 0, 1, and 2, have been placed in an Excel-based questionnaire. In this organization’s assessment, the questions/statements were addressed during meetings with the software maintenance resource and senior management. These meetings also helped in the exchange of information and to raise awareness of the model’s practices proposed at each level. Once compiled, the results present the current maintenance program’s strengths and weaknesses, and its corresponding maturity level. The assessment methodology rating notation closely follows the recommendations of the model’s creators. Since the assessor in this case study had no previous experience with the model or the assessment method, the maturity rating procedure was simplified to improve the objectivity of the responses to each question/statement associated with the first two levels of maturity (0 and 1):

For level 0, the admissible responses to the questions/statements are “True” and “False.” All the statements are stated in the negative form to try to confirm the absence, rather than the presence, of a specific software maintenance process. Take, for example, Req1.0.1. The software maintenance organization does not manage user requests or software events. Responding “True” to this question generates a rating of 0%, and “False” a rating of 100%.

For levels 1 and 2, the choice of responses is in conformity with ISO 15504:

N: Not Achieved, 0–15%. There is no or little evidence that the process objectives and goals have been met.

- P: Partially Achieved, 16%–50%. Some of the objectives and goals of the process have been met.
- L: Largely Achieved, 51%–85%. A significant portion of the objectives and goals of the process have been met.
- F: Fully Achieved, 86%–100%. The objectives and goals of the process have been fully met.

To facilitate the calculation of the percentages and to reduce possible subjectivity due to the lack of experience of the assessor, the value 0% was assigned if the process was not carried out, or “Not Achieved.” For the other rating levels (P, L, and F), the median value was used, giving the following four possible ratings:

N: Not Achieved (0%)

P: Partially Achieved  $(50\%-16\%)/2 + 16\% = 33\%$

L: Largely Achieved  $(85\%-51\%)/2 + 51\% = 68\%$

F: Fully Achieved  $(100\%-86\%*)/2 + 86\% = 93\%$

The organization sponsors reviewed and accepted this rating approach.

#### **12.3.4.2 Assessment Results**

This section presents the results of the S3<sup>m®</sup> maturity assessment of the maintenance of this invoicing software.

**Maturity Level 0 Practices Assessment.** Level 0 is the entry point to S3<sup>m®</sup> and provides an overview of each key process area for the four process domains. The assessment results are illustrated in Figure 12.5. It can be observed that this maintenance organization does not consistently fulfill the criteria of each process domain. For the Process Management domain, the organization fulfills only some of the requirements; for instance, software maintenance is performed without a life cycle, and software maintenance work is perceived as a marginal activity.

Furthermore, no software training is planned or provided to the maintainer. There is no monitoring of maintenance process performance and no process measurement. Time sheets are collected, but they are not analyzed or used for improving software maintenance. Of course, there is no measurement for the purpose of innovation in the current software maintenance activities and technologies.

In contrast, this organization is interested in this process assessment and, therefore, meets the criteria of practices 5.0.2, Study of a Solution for Improvement, and 5.0.3, Impact of an Improvement, contained in the KPA Maintenance Innovation and Deployment.

With an overall rating of only 29%, this process domain is not under control, and this has a definitive impact on the quality of the service given to customers.

\*The 86% value represents the minimum for the “Fully Achieved” rating in ISO 15504.

These findings have raised awareness at the management level that action is required in the short term.

The second process domain of the S3<sup>m®</sup> model, Event/Request Management, was also assessed at level 0, since no corresponding process is implemented by this organization; there is no process for managing user requests, all interactions being conducted informally between the users and the maintainers. For instance, users phone the maintainer directly as soon as they encounter a problem with the software, and no trace of this event is kept. The maintainer does his best to handle the most urgent situations. Maintenance is performed based on the availability of resources, and, if the maintainer is on sick leave or is assigned to another task, maintenance is postponed. In contrast, if the software monthly cycle cannot be executed at the end of the month, senior management assigns it top-priority maintenance status. In summary, none of the KPAs in this domain are performed in this organization.

The third process domain, Evolution Engineering, contains the basic analysis and programming activities. This process domain of the S3<sup>m®</sup> model is rarely missed if some level of maintenance is performed, even if it is fairly basic. Each of the process areas is carried out in this organization. Through interviews, it was established that having a single maintainer assigned to development and maintenance helps in this situation. Transition and predelivery activities are conducted for each phase of the project by the personnel responsible for the software. Impact analysis, programming, and tests are validated with the users. Support is given to users after corrections and modifications have been completed. Users are also informed by e-mail of the dates and times of failures and interruptions in the running of the software. Unfortunately, these e-mailed failure data are not included in monthly reports of failures and downtime. The employee assigned to maintenance tasks also bases his reports on the same fundamental processes, and stresses that there is little documentation in the source code and that the validations and internal controls are limited. This process domain meets all criteria for level 0.

In the fourth process domain, Support to Evolution Engineering, the only process area that meets the criteria is the KPA, Configuration and Version Management; the maintainer has developed a folder structure to classify the source versions. This very basic configuration management technique works well when there is only one maintainer. None of the other process areas of this fourth domain is executed, since no independent quality assurance activity is conducted on the maintainer's work and there is no documented software quality guidance in the corporate quality system. The individual in charge of corporate quality assurance does not carry out any audit in this area of the organization. The analysis of the causes of failure does not follow a problem resolution process, this being left to the discretion of the maintainer. Lastly, there is no plan for rejuvenating the software under maintenance. In summary, only 20% of the criteria in this process domain are met.

The aggregated rating of the four process domains is 37%, which corresponds only to the "Partially Achieved," 16%–50% scale level for level 0 of maintenance maturity in S3<sup>m®</sup> (see Figure 12.5)

Process Domain	Process Area	Level 0 Question	Rating	% Completed
Process management	Maintenance process focus	1.0.1	Yes	0%
	Maintenance process/service definition	2.0.1	Yes	0%
	Maintenance training	3.0.1	Yes	0%
	Maintenance process performance	4.0.1	Yes	0%
	Maintenance innovation and deployment	5.0.1	Yes	0%
		5.0.2	No	100%
		5.0.3	No	100%
Total				29%
Event/request management	Event/request management	1.0.1	Yes	0%
	Maintenance planning	2.0.1	Yes	0%
	Request/software monitoring and control	3.0.1	Yes	0%
	SLA and supplier agreements management	4.0.1	Yes	0%
Total				0%
Evolution Engineering	Predelivery and transition services	1.0.1	No	100%
	Operational support services	2.0.1	No	100%
	Software evolution and correction services	3.0.1	No	100%
	Verification and validation	4.0.1	No	100%
Total				100%
Support Evolution Engineering	Configuration and version management	1.0.1	No	100%
	Process, service and software quality assurance	2.0.1	Yes	0%
	Maintenance measurement and analysis	3.0.1	Yes	0%
	Causal analysis and problem resolution	4.0.1	Yes	0%
	Software rejuvenation, migration and retirement	5.0.1	Yes	0%
Total				20%
Level 0 rating				37%

Figure 12.5 Summary of level 0 assessment results [Paquette 2006].

**Maturity Levels 1 and 2 Practices Assessment.** Even though the level 0 scale rating is well below the 86% passing target, there is still interest in assessing practices at levels 1 and 2.

At maturity level 1, the process applied in the organization is characterized as the work of the individual maintainer who carries out the software maintenance. This characterization maps well to what is happening in this specific organization. For the first process area, some improvement work is being carried out.

The rating given in response to the practice Pro1.1.1, *Rate how informal the improvement of software maintenance processes is*, is “Largely Achieved,” because some key improvements are being made. Technical improvements have also been

reported, consisting in this organization of the implementation of programming rules and file structures.

On that basis, the statement Pro1.1.2, *Some individual improvement initiatives aim mostly at improving technical aspects of software maintenance processes*, is rated by the assessor as “Fully Achieved.” The maintenance and service support processes are also based on the expertise and initiatives of the individual maintainer. Some personal notes and embryonic processes have been initiated on an ad hoc basis by the individual maintainer.

Although the individual initiatives were ad hoc, the assessor rated the questions referring to them—Pro2.1.1, *Are the maintenance processes/services informal and based on the experience of individuals?* and Pro2.1.2, *Are individual initiatives, for defining maintenance processes/services, mainly trying to address technical aspects of the software or describe, in a local format, the activities of a specific maintenance organization?*—and the statement, Pro5.1.2, *Individual initiatives, for improvement and innovation, target mainly the technical aspects of software maintenance, as accurately representing what is done in this organization, even though there are few formal artifacts.*

This process assessment activity contributed to rating question Pro5.1.3, *Are assessments of new processes, technologies, methodologies, and tools for maintenance are performed informally?* as “Largely Achieved” in this case.

The next process domain, Evolution Engineering, is again assessed as quite strong. All process areas were rated as “Fully Achieved” and with the highest rating of 93%. In this specific context, the developer is also the maintainer; transition is therefore not an issue and is awarded full marks. Support, although informal, is offered by the individual who developed the initial software; when a failure occurs, the users can phone the maintainer or walk to his desk. All maintenance services are executed using the maintainer’s ad hoc processes. Verification and validation are performed informally after each change in production; the timing varies based on the availability of the maintainer. All level 1 questions for this process domain reflect the situation of this “one-person” group.

Of the KPAs of the process domain Support to Evolution Engineering, only one is executed. Again, configuration management is carried out by a single person. This person checks and validates his or her own work, being the only one with the required expertise. All the other level 1 practices are rated as “Not Achieved.” The overall rating for maturity level 1 is 36%, that is, “Partially Achieved.”

Figure 12.6 presents the results of the practices that rated higher than “Not Achieved” for level 1 of S3<sup>m®</sup>.

The assessment for maturity level 2 identified only two process areas with some processes rated higher than “Not Achieved.” These two process areas fall into the Evolution Engineering process domain; ad hoc reporting and data extractions are performed by the maintainer and are rated “Largely Achieved,” whereas evolutions and corrections are made in the original programming language throughout the maintenance.

Figure 12.7 shows that only the following two practices have been rated higher than “Not Achieved” for level 2 of S3<sup>m®</sup>.

Process Domain	Process Area	Level 1 Question	Rating	% Completed
Process management	Maintenance process focus	1.1.1	L	68%
		1.1.2	F	93%
	Maintenance process/service definition	2.1.1	L	68%
		2.1.2	L	68%
	Maintenance innovation and deployment	5.1.2	L	68%
		5.1.3	L	68%
Total				36%
Evolution Engineering	Predelivery and transition services	1.1.1	F	93%
	Operational support services	2.1.1	F	93%
	Software evolution and correction services	3.1.1	F	93%
	Verification and validation	4.1.1	F	93%
Total				93%
Support Evolution Engineering	Configuration and version management	1.1.1	F	93%
Total				15.5%
Level 2 rating				36%

L: Largely Achieved F: Fully Achieved

**Figure 12.6** Summary of the level 1 assessment results (first three process domains) [Paquette 2006].

### 12.3.4.3 Summary of Case Study

In the organization being assessed, the S3<sup>m®</sup> assessment has made it possible to better understand the current work status of the maintenance of a single software application, as well as the maintenance issues arising from some of the failures of the development process. For instance, pressures from senior management had led to a quick and functional project, but with very weak documentation by the only person assigned to it. This person was initially assigned to the maintenance of this software, but left some time later. After his departure, the process weaknesses became

Process Domain	Process Area	Roadmap	Level 2 Question	Rating	% Completed
Evolution Engineering	Operational support services	Ad hoc requests/reports/services	2.2.6	L	68%
	Software evolution and correction services	Evolution/ Correction	3.2.5	F	93%

L: Largely Achieved F: Fully Achieved

**Figure 12.7** Summary of level 2 assessment results (fourth process domain) [Paquette 2006].

more serious and obvious. For instance, the new person assigned to the maintenance of this software did not receive training on the software, and had only limited documentation at his disposal. Maintenance activity became hectic for software that was critical to the business operations. Under these conditions, management focus was required and the S3<sup>m®</sup> was used to obtain a diagnosis by evaluating the maturity level of the maintenance of this software.

The results of this evaluation showed that this software maintenance organization did not meet the requirements of level 0 of the S3<sup>m®</sup> maturity model for the invoice application. Although the Evolution Engineering process domain was assessed at level 1, and a KPA at level 2, two of the other three process domains did not even receive the level 0 rating of “Largely Achieved” (i.e., within the 51%–85% range).

Following presentation of the assessment results, several e-mails circulated at management level. Management is now aware of the benchmarking results with the S3<sup>m®</sup> model, which provided documentary evidence that their maintenance processes for this software under maintenance were creeping along at level 0 on the maturity scale! To address this issue in a satisfactory manner, various steps had to be taken.

Various recommendations were tabled before the management team to improve their maintenance, based on the S3<sup>m®</sup> model, which, if accepted, could constitute the first step in a plan to improve the organization’s maintenance maturity level. The key to success in implementing them lies in senior management commitment and continuous support for the action items proposed, which are the following:

First, it is of primary importance to have senior management’s support in order to officially recognize the importance of software maintenance within the organization.

Second, an improvement project for software maintenance must be initiated and supported with appropriate budget and resources.

Third, documentation and prioritization of the multiple and sometimes concurrent maintenance requests submitted to the maintainer is recommended to ensure that management is aware of, and understands, the amount of effort involved.

Fourth, the maintenance manager is to prepare a list of all the possible short-term improvements that can be implemented.

Fifth, improvement activities based on the S3<sup>m®</sup> are to be assigned to different individuals based on their expertise. For instance, process areas involving process and management are to be assigned to managers, and more technical process areas to programmers.

The final action item recommended is to conduct another maturity assessment to monitor how the organization is progressing.

Assessing software maintenance using S3<sup>m®</sup> enabled this organization to better identify and understand their current process weaknesses and to present a road map for change to their management team.

## 12.4 SUMMARY

The evaluation activities described in this chapter were focused on identifying the usefulness of the proposed maturity model for identifying the current maturity level, and help to determine what could be done next to improve the situation. This was demonstrated in four case studies that address different areas of the process model. The feedback from the trials was used to produce S3<sup>m®</sup> version 2.0.

# Chapter 13

---

## Summary

This book has presented a software maintenance maturity model to help maintainers identify their process maturity level and guide them to higher maturity processes. The organizations interested in improving software maintenance activities are likely to have an improvement program already under way. This maintenance maturity model was developed to address the uniqueness of software maintenance.

This software maintenance maturity model ( $S3^{m^{\circledR}}$ ) was developed to assess the maturity of the software maintenance processes and to help in orienting improvement activities. The proposed  $S3^{m^{\circledR}}$  maturity model has been developed as a complement to CMMi and ISO 15504. The primary goal of the proposed maturity model is to support the improvement of software maintainers involved in small maintenance activities. The identification of key differences between the development and maintenance functions was based on industry experience, international standards, and the literature on software maintenance.

The first chapter presented an overview of key maintenance issues, including an inventory of activities/processes and practices. It also discussed how measurement can play an important role in assessing process and product quality.

Chapter 2 presented an overview of the literature on software engineering maturity models, including maturity model architecture, development steps, and validation approaches. Available models addressing various maintenance processes were identified and used to enhance the detailed inventory of activities/processes and practices.

The  $S3^{m^{\circledR}}$  model architecture, design decisions, and detailed processes, as well as its theoretical formalization, were presented in Section 3.4. To illustrate the detailed content of the  $S3^{m^{\circledR}}$ , the goals and objectives of each KPA, together with the detailed practices from maturity levels 0, 1, and 2, were presented in subsequent Chapters 4 to 11.

In Chapter 12, four case studies were presented in which the proposed maturity model was used for software maintenance process improvement. In Appendix A, proposals are presented to enhance existing software maintenance international standards process models.

In the remainder of this chapter, some maintenance key issues are revisited, together with a summary on how this maturity model is contributing solutions.

## 13.1 THE MAINTENANCE ISSUES REVISITED

How unique are software maintenance processes and can software maintenance processes be improved?

1. Is software maintenance a specific domain of software engineering? If so, what are its specific processes and activities?
2. Are the unique processes of software maintenance well reflected in the current international standards?
3. Is there an existing maturity model proposal that covers the entire set of software maintenance unique activities? Does the CMMi adequately address the specific processes and activities of software maintenance?
4. What would be the proposed architecture of a capability maturity model that would address the entire set of unique software maintenance activities?
5. How can such a model be used in practice to support the improvement of software maintenance?

Questions 1 and 2 consider whether or not software maintenance is different from other software engineering domains, due to its specific, unique activities and whether or not it is well represented in international standards. Question 3 investigates whether or not there is already a maturity model proposal that entirely covers the software maintenance domain. Question 4 asks what this maturity model would look like. Question 5 asks how such a model could help practitioners improve their software maintenance processes.

## 13.2 QUESTIONS 1 AND 2—IS MAINTENANCE A SPECIFIC DOMAIN OF SOFTWARE ENGINEERING?

The literature confirms that software maintenance is a specific domain of software engineering. Further confirmation comes from the inclusion of software maintenance in the SWEBOK Guide, ISO TR 19759 [Abran et al. 2005]. Indeed, the findings of this research were used to enhance the content of the text of the SWEBOK during the 2004 review of the maintenance chapter. The SWEBOK confirms and presents details of the software maintenance knowledge that is accepted as specific domain knowledge of software engineering.

To answer the second question, the software maintenance standard (ISO 14764) was studied. It was confirmed that software maintenance and software development have indeed some activities in common. Maintenance refers as well to specific software development activities, listed clearly in international standards. The standards also indicate that maintainers must make sure to adapt them to the specific maintenance context and not just use them as the developers intended.

Many of the unique processes and activities that were inventoried for the S3<sup>m®</sup> model are not yet part of the current international software maintenance standards.

### **13.3 QUESTION 3—DOES THE CMMI ADEQUATELY ADDRESS SOFTWARE MAINTENANCE?**

CMMi does not cover software maintenance adequately. Its primary focus is on software projects using a project management approach and it does not address the unique processes and activities of software maintenance. Similarly, the literature review did not reveal any comprehensive diagnostic techniques for evaluating the quality of the unique maintenance processes and activities; none of previous work covers the entire set of activities included in the process model introduced here. It should be noted, however, that the S3<sup>m®</sup> model includes important contributions from CMMi, and from the Kajko-Mattsson, Niessink, ITIL, and CobIT models that have been mapped into our S3<sup>m®</sup> model.

### **13.4 QUESTION 4—WHAT WOULD THE ARCHITECTURE OF A SOFTWARE MAINTENANCE MATURITY MODEL LOOK LIKE?**

This book presented an ISO 12207-like process model in conjunction with the CMMi architecture to create an original maturity model specific to the maintainers' unique processes. The alignment with the CMMi architecture is considered beneficial, as many improvement initiatives already use this model and its assessment techniques. This book described the proposed model architecture and its maturity scales, and provided examples, for each maturity level, of detailed practices for maturity levels 0, 1, and 2.

### **13.5 QUESTION 5—HOW CAN SUCH A MODEL BE USED IN PRACTICE?**

The steps taken in the V&V activities described in Chapter 12 were meant to test the proposed maturity model in an industrial setting. The trials were aimed at identifying the usefulness of the proposed maturity model and its ability to assess maturity, and to aid in improvement initiatives. This has been documented in case studies addressing different areas of the process model. Users of the model in the trials confirmed that the notion of maturity presented by the model reflected the state of the organization assessed. It has been argued that the model practices were useful when used to improve software maintenance processes.

## **13.6 LESSONS LEARNED AND CONTRIBUTIONS**

The main lessons learned and contributions described in this book are:

1. The current representation of software maintenance in international standards (i.e., ISO 12207 and ISO 14764) only partially covers the activities of software maintenance as experienced by practitioners.

2. Up to now, there have been no maturity model proposals covering the entire set of unique activities of software maintenance, as described by the SWE-BOK.
3. A maturity model architecture and structure to reflect the uniqueness of the software maintenance domain.
4. An original model and detailed practices provided to structure the software maintenance activities in progressively more challenging levels of maturity.
5. For validation and illustrative purposes, highlights of case studies in industrial contexts.

Some of the content of this S3<sup>m®</sup> model were included in the 2004–2005 version of the SWEBOK chapter dedicated to software maintenance [Abran et al. 2005].

## 13.7 FURTHER READING

Readers interested in further insights into the model will find in the appendices more detailed information:

*Appendix A—Maintenance Standards Models and Enhancement Proposal.* This appendix presents an overview of the models of various maintenance models found in the standards, as well as some of the views of the authors of this book on how to improve them.

*Appendix B—Term Assignments for Students.* This appendix presents a set of 31 term assignments dealing with various maintenance questions that students are invited to select and investigate over the course of a term. Some of these assignments can be addressed by analyzing the literature on software maintenance, whereas for others, students will have to interact with industry to complete their assignments.

*Appendix C—Acronyms and Glossary.* A list of acronyms and a glossary of almost 400 maintenance-related terms used in this book and in the S3<sup>m®</sup> model. Many of these terms are based on international standards.

# Appendix A

---

## Maintenance Standards Models and Enhancement Proposal

### A.1 SOFTWARE MAINTENANCE STANDARDS

What about the many unique processes that were identified earlier, but which are not yet included in the software maintenance standards? This literature review identifies a large number of unique software maintenance activities that have not yet been addressed in the international standards. Included is a list of unique maintenance activities:

- Annual planning and management of maintenance [Pigoski 1994, Zitouni and Abran 1996]
- Management of maintenance employees [Zitouni and Abran 1996]
- Management of customer service agreements and their roles [Mueller 1994, April 2001, Bouman 1999, McBride 1990, McBride 1996, Karton 2007, IT Governance Institute, 2006]
- Interception/surveillance of application software currently in production (problem prevention) [ITIL 2007b]
- Maintenance and support-specific service measures [Abran 1991, Abran 1993, Stark et al. 1994, McGarry 1995]
- Customer service (problem-response system) in regard to outages, preventive maintenance, and postoutage return to service [April 2001]
- The study of different types of requests for changes handled by a help desk call center and its support software [Bennett 2000]
- Activities to evaluate the impact of a change [Bennett 2000]
- Test specialization and regression verification [Bennett 2000]

- Investigation and answering questions concerning business rules of application software [Pressman 2001, April 2001]
- Unique process of approval/refusal of work for application software MRs, according to their size [April 2001]
- 24/7 scheduled management for operations support and escalation in case of problems [ITIL 2007d, McBride 1995]
- Management of the personnel interface and the role of operations (change management, calls concerning a system failure, environment and data recovery after a disaster, data recovery and restart of work, processing time investigations, automated scheduling, backups, and management of disk space and tape libraries) [ITIL 2007b]
- Management of subcontracting, maintenance and support contracts, licensing, escrow delivery, and outsourcing [Carey 1994, April 2001, McCracken 2002, Barry 2002]
- Obligation to improve the application software portfolio's performance [Zitouni and Abran 1996]

Since international standards only include activities that are generally accepted at the time they are published, it is normal that a 1998 standard will not include all maintenance activities generally accepted in the industry in 2005. This list was submitted to the ISO committee in charge of preparing a new version of the ISO14764 standard, through the Canadian national commenting body process. The committee's reply was as follows:

This comment is out of scope with the NWI which is to merge the content of the IEEE 1219 into ISO14764. This comment will be placed in a "deferred comment database" for the future revision. This revision will be started by an NWI proposal, subject to approval by SC7 and JTC 1.

This answer provides clarification that the committee was interested in this list, but was focused on merging the IEEE 1219 and ISO 14764 standards before they can undertake new items of work.

According to Bennett [Bennett 2000, s9.3], the ISO software maintenance standards (i.e., ISO 14764) are oriented toward "classic" activities and do not address certain aspects of the processes that support software maintenance. Consequently, Bennett stipulates that the current standardized software maintenance process models approximately correspond to level 2 SEI practices [SEI 2001]. In fact, this is where they have been assigned in the standards pyramid in Figure 1.8.

In conclusion, there are a large number of currently published software standards that include software maintenance topics. The 2006 version of ISO 14764 is the current software maintenance standard. The maintenance domain refers to software development standards and two of its support processes. Maintainers must, therefore, use software development standards while making sure to adapt them to the

specific context of maintenance [ISO 1998, s8.3.2.1 and s8.3.2.2]. Incidentally, by nature, standards do not include subjects on emerging technologies. The list of unique software maintenance activities provided to the ISO is a candidate for inclusion in a future version of the standard.

There is also a proposal that a relationship exists between application software criticality, size, the process maturity of an organization, and the external process requirements and the organization's progressive adoption of standards.

From the observations documented in the previous section, we suggest that the content of ISO 14764, which is aligned with ISO 12207, be updated to reflect a newer perspective of software maintenance processes and activities. Figure 3.4 was created for proposed updates to the maintenance topics of both ISO 14764 and ISO 12207.

In Figure A.1, the updated process view of software maintenance is highlighted, showing how it could be integrated into the existing process model of those international standards.

## 5 Primary life-cycle processes

### 5.1 Acquisition process

### 5.2 Supply process

### 5.3 Development process

### 5.4 Operation process

### 5.5 Maintenance process

- Predelivery and Transition
- Event/Request Management
- Operational support services
- Correction services
- Evolution services
- Version management
- Rejuvenation, migration and retirement

## 6 Supporting life-cycle processes

### 6.1 Documentation

### 6.2 Configuration Management

### 6.3 Quality Assurance

### 6.4 Verification

### 6.5 Validation

### 6.6 Joint Review

### 6.7 Audit

### 6.8 Problem Resolution

## 7 Organizational life-cycle processes

### 7.1 Management

### 7.2 Infrastructure

### 7.3 Improvement

### 7.4 Training

**Figure A.1** Proposed update to ISO 12207 maintenance processes.

# Appendix B

---

## Term Assignments for Students

**Term assignment #1.** Identify the books and papers that discuss software maintenance and have been published since 2005. Identify, in these documents, software maintenance topics and activities that could contribute to updating the upcoming version of the maintenance chapter of the *Guide to the SWEBOk* [Abran et al. 2005] (ISO 19759) and to the ISO 14764 standard.

**Term assignment #2.** Propose to an organization, and test, a maintenance standards maturity model. Document the lessons learned about this theoretical model based on your observations in industry.

**Term assignment #3.** Identify the organizational model used, for maintenance, in your organization. Discuss with the maintenance manager the reasons for this choice of organizational model and document its pros and cons.

**Term assignment #4.** Identify and present to your organization the maintenance related international standards that could be useful. Document the proposals and the feedback from participants (pros and cons).

**Term assignment #5.** Propose an extension to one of the three proposals of software maintenance ontologies by including all the topics tackled in the maintenance chapter of the *Guide to the SWEBOk*. As a starting point and for guidance, use the proposals put forward by Kitchenham [1999] or Ruiz [2004].

**Term assignment #6.** Propose an update to the ISO 14764 standard. To do this, you must use the comment template and the formal ISO/IEC/JTC1 process to comment on the current standard.

**Term assignment #7.** Compare the activities in the CMMi with the activities found in Term assignment #2. It is claimed that the CMMi addresses software maintenance. Document the CMMi viewpoint on software maintenance.

**Term assignment #8.** Carry out a field study of a software maintenance organization (with over 50 maintainers and customers) to identify both internal and external software maintenance problems. Then discuss your findings by comparing them with those of the Dekleva survey.

**Term assignment #9.** Make a thorough review of the maintenance measurement literature and present a measurement model of the processes specifically designed for software maintainers. Work out the strengths and weaknesses of your proposal.

**Term assignment #10.** Assess the quality of software maintenance documents. Analyze the maintainability of a piece of software by using a software source code assessment tool (for example, Checkstyle, Logiscope, etc.). Identify which of the ISO 9126 maintainability measures can be assessed.

**Term assignment #11.** Analyze the maintenance service agreement of your organization. Propose improvements to it. If you do not have access to an industrial site, develop a maintenance service agreement maturity model of your own.

**Term assignment #12.** Evaluate the software maintenance contracts of your organization. To do this, do a literature review of this topic and identify the ideal contract (from the point of view of the customer). Next, analyze the contract and identify the 10 items, in order of importance, that should be discussed in the forthcoming negotiations with the supplier. Give your customer ammunition!

**Term assignment #13.** Investigate further the topic of software maintenance benchmarking.

- A) If you have access to a maintenance benchmarking study carried out by a consultant: Describe the benchmarking approach, the validity of the data, and the conclusions drawn by that study. Analyze the results of the benchmarking study and discuss the feedback of the software maintainers on the study recommendations.
- B) If you do not have access to a benchmarking study carried out by a consultant but you can conduct a benchmarking test with some maintenance requests, use the ISBSG benchmarking process with some requests and submit your data to the ISBSG; do a benchmarking study yourself, using both the ISBSG data and your own data.
- C) If you do not have access to a benchmarking study carried out by a consultant and if you do not have access to an industrial site to carry out a benchmarking test, do a literature review on benchmarking and develop a software maintenance maturity model of your own.

**Term assignment #14.** Take part in an ISO 9001 certification of a software maintenance group. If this is not feasible, identify in the ISO 90003 Guide to ISO 9001 the items that must be covered by certification and propose a certification plan.

**Term assignment #15.** Develop an evaluation method for the software maintenance maturity model.

**Term assignment #16.** Analyze the ISO 15504 standard and use it to improve the evaluation support software ( $S3^m^{\circledR}Assess$ ).

**Term assignment #17.** Analyze the ISO 15504 standard and identify the improvements necessary to claim that the software maintenance maturity model conforms to the recommendations of this international standard. Carry out the necessary improvements.

**Term assignment #18.** Carry out a new literature review of software maturity models and update the model inventory.

**Term assignment #19.** Carry out a survey to update the Parikh study on the evolution of maintenance practices. Compare your survey results with those of the Parikh study.

**Term assignment #20.** Identify other candidate sources of good practices for the maturity model. Explain the relevance of these new sources.

**Term assignment #21.** Analyze the key areas that were selected for the architecture of the software maintenance maturity model.

**Term assignment #22.** Study the Sarbane–Oxley law (or other equivalent nationally imposed law) and discuss how this new mandatory legal requirement can be integrated into the maturity model.

**Term assignment #23.** Study the criteria that a lead appraiser must meet to use the software maintenance maturity model. Develop a training guide for a new appraiser.

**Term assignment #24.** Using the CMMi training guide as a reference, develop a training course to prepare a team to carry out an evaluation of the software maintenance processes. Base the course on the model suggested in this book. Clearly identify the material necessary for the evaluation.

**Term assignment #25.** Design and carry out a survey (questionnaire and interviews) on the factors underlying the success of firms specializing in software improvement. After you have collected the data, rank those factors in order of importance and comment on what specialists do to ensure the success of this type of survey.

**Term assignment #26.** Make an inventory of problems related to the use of maturity models. Design and carry out a survey (questionnaire and interviews) on the

problems encountered by organizations specializing in software improvement. Identify the problems that are no longer relevant, and the new ones that have recently come up.

**Term assignment #27.** Identify the generic practices of software maintenance.

**Term assignment #28.** Review the text that describes each maturity level and identify inconsistency and possible improvements.

**Term assignment #29.** Review the model road maps and identify new ones that have not yet been identified. Justify their importance and why we would need to add this new perspective. Take examples from your organization.

**Term assignment #30.** Continue to evolve and improve the knowledge-based system ( $S3^{m^{\circ}dss}$ ) that supports the software maintenance maturity model ([http://www.gelog.etsmtl.ca/S3<sup>m<sup>°</sup>DSS</sup>/](http://www.gelog.etsmtl.ca/S3^{m^{\circ}}DSS/)).

**Term assignment #31.** Review the model event/request management KPA to add some level 2 practices.

# Appendix C

---

## Acronyms and Glossary

### ACRONYMS

4GL	fourth-generation language
ASQC	American Society for Quality Control
CASE	computer-aided software/system engineering
CI	continuous improvement
CM	configuration management
CMM	Capability Maturity Model
CMMi	Capability Maturity Model, Integrated
COCOMO	Constructive Cost Model
DP	defect prevention
ERD	entity-relationship diagram
FMEA	failure mode and effect criticality analysis
FSM	functional size measurement
FTE	full-time equivalent
GPC	general purpose computer
HLL	high-level language
HMI	human-machine interface
HR	Human Resources
H/W	hardware
IC	intergroup coordination
IEEE	Institute of Electrical and Electronics Engineers
IPD	integrated product development
IPT	integrated product team
IS	information system
ISBG	International Software Benchmarking Standards Group
ISM	integrated software management
ISO	International Organization for Standardization, Geneva
ISO/IEC	joint ISO and IEC work
IT	information technology
Kloc	thousands of lines of source code
LCC	life-cycle cost
Mngmt	management

MIS	management information system
Mloc	millions of lines of source code
MR	modification request
OO	object oriented
OOA	object-oriented analysis
OOD	object-oriented design
OPD	organization process definition
OPF	organization process focus
OS	operating system
OSS	operation support system
PCM	process change management
PCTE	portable common tool environment
PM&A	process measurement and analysis PR (peer review)
PR	problem report
PROM	programmable read-only memory
QAI	Quality Assurance Institute
QC	quality control
QE	quality engineering
QPM	quantitative process management
S3 <sup>m</sup>	Software Maintenance Maturity Model
SA/SD	structured analysis/structured design
SCM	software configuration management
SDL	specification design language
SEI	Software Engineering Institute
SEPG	Software Engineering Process Group
SLA	service level agreement
SPE	software product engineering
SPP	software project planning
SPT&O	software project tracking and oversight
SQA	software quality assurance
SQM	software quality management
SSM	software subcontract management
S/W	software
SWEBOK	Software Engineering Body of Knowledge, public (IEEE, <a href="http://www.swebok.org">www.swebok.org</a> )
TCM	technology change management
TP	training program
V&V	verification and validation

## **GLOSSARY**

### **Definitions**

Terms used in the S3<sup>m</sup>® model are accepted industry wide. When preparing a S3<sup>m</sup>® evaluation, it may be necessary that the terminology be amended to the organization

culture. Do not hesitate to do this but use your professional judgement to preserve the integrity of the terms. In general the following order of precedence has been observed:

- The terminology of the Camélia model
- The terminology of the Abran–Zitouni model
- ISO/IEC JTC1/SC7 vocabulary
- ISO 15504 vocabulary
- ISO/IEC 14764 vocabulary
- Other sources (CMMi, COBIT, and ITIL)

**Acceptance criteria** The criteria that a system or component must satisfy in order to be accepted by a user, customer, or other authorized entity.

**Acceptance testing** Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system (IEEE STD 610).

**Acquirer** A party, commonly an organization, that acquires or procures a system or a component of the system from a supplier.

**Acquirer–supplier agreement** An arrangement between two parties (an acquirer and a supplier) that defines the tasks to be performed and the acceptance criteria to be applied to delivered items.

**Acquisition** The process of obtaining something through a contract.

**Acquisition organization** That entity which has the oversight responsibility for the software acquisition project and which may have purview over the acquisition activities of a number of projects or contract actions.

**Acquisition organization's standard software acquisition process** The acquisition organization's fundamental software acquisition process that guides the establishment of each project's defined software acquisition process.

**Action item** (1) A unit in a list that has been assigned to an individual or group for disposition. (2) An action proposal that has been accepted.

**Action proposal** A documented suggestion for change to a process or process-related item that will prevent the future occurrence of defects identified as a result of defect prevention activities. (*See also* software process improvement proposal.)

**Activity** Any step taken or function performed, either mental or physical, toward achieving some objective. Activities include all the work the managers and technical staff do to perform the tasks of the project and organization.

**Adaptive maintenance** The modification of a software product performed after delivery to keep a computer program usable in a changed or changing environment. Adaptive maintenance provides enhancements necessary to accommodate changes in the operational or hardware environment. These include changes to implement new system interface requirements, new system requirements, or new hardware requirements.

**Age (of a problem)** Number of days between the day a problem ticket was written and (1) today, if the ticket is still open, or (2) the day the ticket was closed.

**Agreement** The mutual acknowledgement of terms and conditions under which a working relationship is conducted.

**Allocate** Assign performance requirements to a function, process, behavior, or other logical element of the system.

**Allocated baseline** The initially approved documentation describing a subsystem's functional, performance, interoperability, and interface requirements that are allocated from those of the system or a higher-level subsystem; interface requirements with interfacing subsystems; design constraints; derived requirements (functional and performance); and verification requirements and methods to demonstrate the achievement of those requirements and constraints. Generally, there is an allocated baseline for each subsystem to be developed.

**Application domain** A bounded set of related systems (i.e., systems that address a particular type of problem). Development and maintenance in an application domain usually require special skills and/or resources. Examples include payroll and personnel systems, avionics, command and control systems, compilers, and expert systems.

**Appraisal method** The set of steps or procedure for conducting a systems engineering appraisal. The appraisal method consists of five phases: commitment, preparation, on-site, post-appraisal, and appraisal follow-up.

**Appraisal team** A team of experienced engineering professionals who are trained in the appraisal method to perform appraisal.

**Appraise** To evaluate the worth, significance, or status of something.

**Approved modification architecture** The disposition of one or more proposed changes authorizing revision of one or more SCIs. A high-level design that provides decisions made about the problem(s) that the product will solve, component descriptions, relationships between components, and dynamic operation description.

**Artifact** *See* Work product.

**Assessed capability** The output of one or more recent, relevant process assessments conducted in accordance with the provisions of ISO/IEC 15504.

**Assessment constraints** Restrictions placed on the freedom of choice of the assessment team regarding the conduct of the assessment and the use of the assessment outputs.

**Assessment indicator** An objective attribute or characteristic of a practice or work product that supports the judgment of the performance of, or capability of, an implemented process.

**Assessment input** The collection of information required before a process assessment can commence.

**Assessment instrument** A tool or set of tools that is used throughout an assessment to assist the assessor in evaluating the performance or capability of processes and in handling assessment data and recording the assessment results.

**Assessment output** All of the tangible results from an assessment (*see assessment record*).

**Assessment participant** An individual who has responsibilities within the scope of the assessment.

**Assessment purpose** A statement, provided as part of the assessment input, that defines the reason for performing the assessment.

**Assessment record** An orderly, documented collection of that information which is pertinent to the assessment and adds to the understanding and verification of the process profiles generated by the assessment.

**Assessment scope** A definition of the boundaries of the assessment, provided as part of the assessment input, encompassing the organizational limits of the assessment, the processes to be included, and the context within which the processes operate (*see process context*).

**Assessment sponsor** The individual, internal or external to the organization being assessed, who requires the assessment to be performed, and provides financial or other resources to carry it out.

**Associated documentation** *See Work product.*

**Associated processes** Processes that enable one or more end products to be put into service, maintained in service, or disposed of at the end of service.

**Attribute** A characteristic of an item; for example, the item's color, size, or type. A measurable physical or abstract property of an entity.

**Attribute indicator** An assessment indicator that supports the judgment of the extent of achievement of a specific process attribute.

**Attributes (of software)** Characteristics of software, such as reliability, maintainability, portability, and complexity. These characteristics are sometimes referred to as quality attributes.

**Audit** An independent examination of a work product or set of work products performed to assess compliance with specifications, standards, contractual agreements, or other criteria.

**Audit client** Person or organization requesting an audit.

**Audit conclusions** Outcome of an audit decided by the audit team after consideration of all the audit findings.

**Audit criteria** Set of policies, procedures, or requirements against which collected audit evidence is compared.

**Audit evidence** Records, verified statements of fact, or other information relevant to the audit.

**Audit findings** Results of the evaluation of the collected audit evidence against audit criteria.

**Audit program** Set of audits to be carried out during a planned time frame.

**Audit scope** Extent and range of a given audit.

**Audit team** One or more auditors conducting an audit, one of whom is appointed as leader.

**Auditee** Organization being audited.

**Auditor** Person qualified and competent to conduct audits.

**Availability** The percentage of downtime of a software. Typically calculated as (number of days in a month · 24h) – (total outage hours)/(number days in a month · 24h).

**Base practice** A software engineering or management activity that addresses the purpose of a particular process.

**Baseline** A formally approved version of a configuration item, regardless of media, formally designated and fixed at a specific time during the configuration item's life cycle (ISO/IEC 12207). Sometimes, a new baseline is referred to as a new release.

**Baseline configuration management** The establishment of baselines that are formally reviewed and agreed on and serve as the basis for further development. Some software work products, for example, the software design and the code, should have baselines established at predetermined points, and a rigorous change control process should be applied to these items. These baselines provide control and stability when interacting with the customer. (*See also* Baseline management.)

**Baseline management** In configuration management, the application of technical and administrative direction to designate the documents and changes to those documents that formally identify and establish baselines at specific times during the life cycle of a configuration item (IEEE STD 610).

**Benchmark** A standard against which measurements or comparisons can be made.

**Benchmarking** The continuous process of measuring products, services, and practices against the toughest competitors or those companies recognized as industry leaders.

**Boundary** A conceptual interface between the software under study and its users.

**Capability** Ability of an organization, system or process to realize a product that fulfils the requirements for that product.

**Capability dimension** The set of process attributes comprising the capability aspects of the reference model of processes and process capability.

**Capability evaluation** An independent process appraisal by a trained team of professionals.

**Capability level** The extent to which the organization can potentially accomplish the essential elements of software engineering as defined in the context of a maturity model. Capability involves the attributes of people, technology, and process. Capability levels are an ascending scale progressing from Initial (Level 0), to Performing, to Managed, to Defined, to Measured, through Optimizing (Level 5).

**Capability maturity model** A description of the stages through which organizations evolve as they define, implement, measure, control, and improve their processes. The model provides a guide for selecting process improvement strategies by facilitating the determination of current process capabilities and the identification of the issues most critical to quality and process improvement.

**Causal analysis** The analysis of defects to determine their underlying root cause.

**Causal analysis meeting** A meeting, conducted after completing a specific task, to analyze defects uncovered during the performance of that task.

**Cause** The origin of a defect or failure as confirmed by analysis.

**Change agent** An individual or group that has sponsorship and is responsible for implementing or facilitating change. An example of a change agent is the software engineering process group. Contrast with change advocate.

**Change authority** The organization or individual appointed to decide on priorities and accept individual change requests.

**Change control** The review, approval/disapproval, implementation, tracking, closure, and status reporting of proposed changes to an item (change management).

**Change management** The process of evaluating the impact of a requirement or design change on the system, analyzing the effects of a proposed change in terms of the system foundation architecture, performance, costs, and schedule criteria. Change management must be supported by configuration management to ensure that decisions to adopt a change in requirement, design, or implementation are reflected in system documentation, engineering drawings, or other representations of the system.

**Characteristic** Distinguishing feature.

**Commitment** A pact that is freely assumed, visible, and expected to be kept by all parties.

**Common cause (of a defect)** A cause of a defect that is inherently part of a process or system. Common causes affect every outcome of the process and everyone working in the process. (See Special cause for contrast.)

**Common cause of variation** Cause of natural variation inherent in a process or system. Removing common causes of variation involves making changes to the process itself. These causes are usually minor and do not cause a process to go out of control. An example is wear and tear on equipment causing greater tolerance variation in an output, such as a drink filler at a fast-food restaurant.

**Common features** The subdivision categories of the CMM key process areas. The common features are attributes that indicate whether the implementation and institutionalization of a key process area is effective, repeatable, and lasting. The CMM common features are the following:

- Commitment to perform—The actions the organization must take to ensure that the process is established and will endure. Commitment to perform typically involves establishing organizational policies and senior management sponsorship.
- Ability to perform—The preconditions that must exist in the project or organization to implement the software process competently. Ability to perform typically involves resources, organizational structures, and training.
- Activities performed—A description of the roles and procedures necessary to

implement a key process area. Activities performed typically involve establishing plans and procedures, performing the work, tracking it, and taking corrective actions as necessary.

- **Measurement and analysis**—A description of the need to measure the process and analyze the measurements. Measurement and analysis typically includes examples of the measurements that could be taken to determine the status and effectiveness of the activities performed.
- **Verifying implementation**—The steps to ensure that the activities are performed in compliance with the process that has been established. Verification typically encompasses reviews and audits by management, and software quality assurance.

**Compatible assessment model** An operational model, used for performing assessments, that meets the defined requirements (for model purpose, scope, elements and indicators, mapping to the reference model, and translation of results) for conformance to the reference model.

**Competent assessor** A person who has demonstrated the necessary skills, competencies, and experience for performing process assessments.

**Completeness** As it applies to requirements, a full consideration of all implications due to higher-level requirements.

**Compliance** Meeting the requirements of a standard or meeting specified requirements.

**Compliance article** An item built, constructed, or coded for the purpose of checking compliance to specified requirements.

**Component** An entity, with discrete structure within a system, that interacts with other components of the system, thereby contributing at its lowest level to the system properties and characteristics. A component can be hardware or software and can be subdivided into other components.

**Composite results** Results that are nonspecific with regard to particular individuals or programs. Typically used to refer to the findings that are presented to the senior management team during the postappraisal phase of an appraisal.

**Concession** Authorization to use or release a product that does not conform to specified requirements.

**Concession; waiver** Written authorization to use or release a quantity of material, components, or stores already produced but which do not conform to the specified requirements.

**Conditional** Information supplied with every package to which it is relevant.

**Configuration** In configuration management, the functional and physical characteristics of hardware or software as set forth in technical documentation or achieved in a product. The arrangement of the parts or elements of a work or deliverable product.

**Configuration baseline** The configuration information formally designated at a specific time during a system's or subsystem's life cycle. Configuration base-

lines, plus approved changes from those baselines, constitute the current configuration information.

**Configuration control** An element of configuration management, consisting of the evaluation, coordination, approval or disapproval, and implementation of changes to configuration items after formal establishment of their configuration identification.

**Configuration identification** An element of configuration management, consisting of selecting the configuration items for a system and recording their functional and physical characteristics in technical documentation.

**Configuration item** An aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process.

**Configuration management** (1) A discipline applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements. Also, technique of applying technical and administrative direction and surveillance to (a) identify and document the functional and physical characteristics of an item, (b) control changes to those characteristics, and (c) record and report change processing and implementation status. (2) A control activity for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design and operational information throughout its life, using disciplined change management.

**Configuration management library system** The tools and procedures to access the contents of the software baseline library.

**Configuration unit** The lowest-level entity of a configuration item or component that can be placed into, and retrieved from, a configuration management library system.

**Conformity** Fulfilment of a requirement.

**Consequence** The outcome of an event or situation expressed qualitatively or quantitatively, being a loss, injury, disadvantage, or gain.

**Consistency** The degree of uniformity, standardization, and freedom from contradiction among the documents or parts of system or component (IEEE STD 610).

**Cosmetic defect** A minor issue that does not prevent the system from being operational but may inconvenience users.

**Constraint** (1) A restriction, limit, or regulation. (2) A type of requirement that is not tradeable against other requirements.

**Constructed capability** A capability constructed from elements of organizational units or of different organizations, that are assembled for the purposes of achieving a particular specified requirement.

**Contingency factor** An adjustment (increase) of a size, cost, or schedule plan to

account for likely underestimates of these parameters due to incomplete specification, inexperience in estimating the application domain, and so on.

**Continuous architecture** A capability model in which the focus areas are all of equal explicit priority to implement, and to which identical generic practices may be added in sets to demonstrate increasing capability. This model contrasts with the staged model, which establishes sets of focus areas that must be performed before other sets in a priority order of increasing maturity.

**Contract** A binding agreement between two parties, especially enforceable by law, or a similar internal agreement wholly within an organization, for the supply of software service or for the supply, development, production, operation, or maintenance of a software product.

**Contract integrity** The adherence and compliance to contractual and legal policies, regulations, and other guidance.

**Contract terms and conditions** The stated legal, financial, and administrative aspects of a contract.

**Contractor** The entity delivering the product or performing the service being acquired, even if that entity is part of the acquiring organization.

**Convertibility** The ability to convert the size obtained by using one measurement method to an accurate representation of the size that would be obtained by using another method.

**Correction** Action taken to eliminate a detected nonconformity.

**Corrective action** Action taken to eliminate the cause of a detected nonconformity or other undesirable situation.

**Corrective maintenance** The reactive modification of a software product performed after delivery to correct discovered problems. The modification repairs code to satisfy requirements.

**Cost** Cost of activities and services, both direct and indirect, involving any negative impact, including money, time, labor, disruption, goodwill, political, and intangible losses.

**Critical computer resource** The parameter of the computing resources deemed to be a source of risk to the project because the potential need for those resources may exceed the amount that is available. Examples include target-computer memory and host-computer disk space.

**Critical technical parameter** A parameter designated in a baseline document as critical to performance.

**Customer** Organization or person that receives a product. Also recipient of a product provided by the supplier. Also the individual or organization that is responsible for accepting the product and authorizing payment to the developing organization.

**Customer dissatisfaction** Customer's opinion of the degree to which a transaction has failed to meet the customer's needs and expectations.

**Customer expectations** What a customer expects to receive from a supplier after the supplier has committed to product or system requirements.

**Customer requirements** The set of essential customer needs, expressed as what the customer wants and why. The requirements outlines the problem that the customer wants to solve. Statements of fact and assumptions that define the expectations of the system in terms of mission or objectives, environments, constraints, and measures of effectiveness. These requirements are defined from a validated needs statement (mission needs statement), from acquisition and program decision documentation, and from mission analyzes of each of the primary system life-cycle functions.

**Customer satisfaction** Customer's opinion of the degree to which a transaction has met the customer's needs and expectations.

**Data** The various forms of documentation required to support a program in all of its areas. Data may take any form (e.g., printed or drawn on various materials, electronic media, or photographs). Data may be deliverable to a customer or nondeliverable for internal use only.

**Data management** Administrative control of program data, both deliverable and nondeliverable. Administrative control involves such items as identification, interpretation of requirements, planning, scheduling, control, archiving, and retrieval of program data.

**Defect** Nonfulfilment of a requirement related to an intended or specified use. Also a flaw or an anomaly found in a system or system component that causes the system or component to fail to perform its required function. In this model, defects are reported on incident reports with four scales: (1) showstopper, (2) serious defect, (3) serious defect with workaround, and (4) cosmetic defect.

**Defect density** The number of defects identified in a product divided by the size of the product component (expressed in standard measurement terms for that product).

**Defect prevention** The activities involved in identifying defects or potential defects and preventing them from being introduced into a product.

**Defect review** A review of a work product, interim or deliverable, that occurs prior to the release of the work product to the next process step. The review involves the creator of the product and subject matter peers, including an outside reviewer for objectivity, who identify defects in the product that would make it unsuitable for use in the next work process, and also develop a common vision of the work product. It is a form of both static testing of the work product earlier than its production and a communication mechanism. *See also Peer review.*

**Defect root cause** The underlying reason (e.g., process deficiency) that allowed a defect to be introduced.

**Defined level** *See Maturity level.*

**Defined process** A repeatable process that has clearly stated inputs, entry criteria, activities, roles, measures, verification steps, outputs, and exit criteria. A defined process is typically defined at the organizational level or tailored from the organization's set of standard processes. Exceptions are documented, reviewed, and approved. A defined process is well characterized and understood, and is described in terms of roles, standards, tools, and methods. A defined process is de-

scribed formally in an organization for use by its managers and practitioners. This description may be contained, for example, in a document or a process asset library. The defined process is what the organization's members are supposed to do. A defined process is developed by tailoring the organizational process to fit the specific characteristics of its intended use. (*See also* Organizational process.)

**Degree of confidence** In this book, the term “degree of confidence” is used only to mean the degree of confidence that software conforms to its requirements.

**Deliverable** An item agreed to be delivered to an acquirer as specified in an acquirer–supplier agreement. This item can be a document, a hardware item, a software item, or any type of work product.

**Delivered products** Those work products that the customer receives. These may also include specifications, interim documents, and prototypes, in addition to the final end product.

**Dependability** Collective term used to describe the availability performance and its influencing factors: reliability performance, maintainability performance, and maintenance support performance.

**Dependency item** A product, action, piece of information, and so on that must be provided by one individual or group to a second individual or group so that the second individual or group can perform a planned task.

**Derived requirement** (1) A requirement that is further refined from a primary source requirement or a higher-level derived requirement. (2) A requirement that results from choosing a specific implementation for a system element.

**Design** The set of decisions about a product that results in a common vision of what need it addresses, and how it addresses or satisfies that need. Typically, a design includes an operational concept (how users are expected or intended to use the product), components and their relationships, and, sometimes, decisions about the processes that will produce, deploy, and support it.

**Design and development** Set of processes that transforms requirements into specified characteristics and into the specification of the product realization process.

**Design authority** The person or organization that is responsible for producing the design of the system.

**Design recovery** Design recovery as a subset of reverse engineering in which domain knowledge, external information, and deduction or fuzzy reasoning are added to the observations of the subject system. The objective of design recovery is to identify meaningful higher-level abstractions beyond those obtained directly by examining the system itself.

**Design review** A formal, documented, comprehensive, and systematic examination of a design to evaluate the design requirements, and the capability of the design to meet these requirements and to identify problems and propose solutions.

**Detailed design baseline** The design solution resulting from detailed design phase activities (e.g., detail drawings, detail specifications, other design solution

descriptions placed under configuration control). Note: This baseline is used to build or construct the compliance articles to be used in the system verification and validation process activities and tasks.

**Developer** An organization that performs development activities (including requirements analysis, design, and testing through acceptance) during the software life-cycle process.

**Development baseline** An agreed upon description of the attributes of a building block that serves as a basis for defining change and qualifying the end products and associated processes recorded at a point in time.

**Development life cycle** A progression from inception to completion of development of a system.

**Development phase** A group of defined activities in a unit of a development life cycle.

**Deviation** A noticeable or marked departure from the appropriate norm, plan, standard, procedure, or variable being reviewed.

**Deviation permit** Authorization to depart from the originally specified requirements of a product prior to realization, for a limited quantity of product or period of time, and for a specific use.

**Direct measure** A measure of an attribute that does not depend upon a measure of any other attribute.

**Direct measurement method** A measurement method designed independently of other measurement methods.

**Discrimination** The ability of a measurement method to react to small changes in the dimensions of the attribute being measured.

**Disposition** Final decision of the personnel responsible concerning an issue.

**Document** A uniquely identified unit of information for human use, such as a report, specification, manual or book. A collection of data, regardless of the medium on which it is recorded, that generally has permanence and can be read by humans or machines.

**Documentation** A collection of one or more related documents.

**Documentation profile** A table of information items describing the content of one or more documents.

**Domain** A subject area that provides some benefit(s) to practitioners. Examples include software or systems engineering, human resources, marketing, finance, facilities construction, renovations and demolition, catering, hospital operating or emergency rooms, and legal or professional advice.

**Effective** Adequate to accomplish the intended purpose.

**Effective process** A series of actions that, when properly performed, produce the intended result (e.g., the desired state change in an object). Given several equally effective processes, their relative efficiency, in terms of resource consumption, can be empirically determined. An effective process can be characterized as practiced, documented, enforced, trained, measured, and able to improve.

**Effectiveness** Measure of the extent to which planned activities are realized and planned results achieved.

**Effectiveness assessment** An analysis of how well a product associated with a design solution will perform or operate given anticipated usage.

**Efficiency** Relationship between the result achieved and the resources used.

**Effort** The number of hours, per person, dedicated to a task.

**Empowerment** The alignment of decision making and its authority, consequences, information, and capability to perform with the goals to be achieved.

**Enabling system** A system, other than the system of interest, that complements the system during its life-cycle stages but does not contribute directly to its functionality; for example, when the system enters the production stage, an (enabling) production system is required.

**End system** The topmost building block in a hierarchy of building blocks. The system that is self contained in terms of its use and operation.

**End user** The individual or group who will use the system for its intended operational use when it is deployed in its environment.

**End user representatives** A selected sample of end users who represent the total population of end users.

**Engineering group** A collection of individuals (both managers and technical staff) representing an engineering discipline. Examples of engineering disciplines include systems engineering, hardware engineering, system test, software engineering, software configuration management, and software quality assurance.

**Engineering plan** The plan for guidance and control of the technical efforts on a program. The engineering plan reflects an integrated technical effort responsible for product development that balances all factors associated with meeting system life-cycle requirements.

**Enhanced capability** A capability greater than current assessed capability, justified by a credible process-improvement program.

**Enterprise** The legal entity within which an organization resides. A unit within a legal entity or spanning several entities, within which one or more programs are managed as a whole. All programs within an enterprise, at the top of the reporting structure, share a common manager and common policies. (*See also Organization.*)

**Environment** (1) The natural conditions (weather, climate, oceanic, terrain, vegetation, dust, etc.) and induced conditions (electromagnetic interference, heat, vibration, etc.) that constrain the design solutions for end products and their enabling products. (2) External factors affecting an organization or program. (3) External factors affecting development tools, methods, or processes. (4) The facilities, hardware, software, firmware, procedures, and documentation needed to develop, test, maintain, and operate software. *See also Software engineering environment and Testing environment.*

**Error prevention analysis** A process that is typically conducted by a working

group of engineering professionals who developed the documentation/product in question. It is an objective assessment of each error, its assigning cause, and the steps to be taken to prevent it. While placing blame is to be avoided, such questions as mistakes, adequacy of education and training, tools capability, and support effectiveness are appropriate areas for analysis.

**Evaluation** The use of reviews, inspections, and/or tests, to determine that a software product or service satisfies specified requirements.

**Evaluation method** A procedure describing the action to be performed by the evaluator in order to obtain the result for the specified measurement or verification applied on the specified product components or on the product as a whole.

**Evaluation module** A package of evaluation technology for a specific software quality characteristic or subcharacteristic.

**Evaluation records** Documented objective evidence of all activities performed and of all results achieved within the evaluation process.

**Evaluation report** The document that presents evaluation results and other information relevant to an evaluation.

**Evaluation requester** The person or organization that requests an evaluation.

**Evaluation technology** A generic term that contains techniques, tools, metrics, measures, and other technical information, used for evaluation (a generic term for the application of techniques, methods, and other technical information).

**Evaluation tool** An instrument that can be used during evaluation to collect data, to perform interpretation of data, or to automate part of the evaluation.

**Evaluator** The organization that performs an evaluation.

**Event** An incident or situation that occurs in a particular place during a particular interval of time.

**Event-driven basis** A review that is performed based on the occurrence of an event within the project (e.g., a formal review or the completion of a life-cycle stage). (See Periodic review for contrast.)

**Event-driven review/activity** A review or activity that is performed based on the occurrence of an event within the project (e.g., a formal review or the completion of a life-cycle stage). (See Periodic review/activity for contrast.)

**Event-tree analysis** A technique that describes the possible range and sequence of the outcomes that may arise from an initiating event.

**Executive** A management role at a higher level in an organization in which the primary focus is the long-term vitality of the organization, rather than short-term program and contractual concerns and pressures.

**Exit criteria** Specific accomplishments or conditions that must be satisfactorily demonstrated before an effort can progress further in the current life-cycle phase or transition to the next phase.

**External attribute** A measurable property of an entity that can only be derived with respect to how it relates to its environment.

**External measure** An indirect measure of a product derived from measures of the behavior of the system of which it is a part.

**External quality** The extent to which a product satisfies stated and implied needs when used under specified conditions.

**Facilitator** Expert in process appraisal responsible for guiding process appraisal team members through a process appraisal activity.

**Facility** The physical means or equipment for facilitating the performance of an action; for example, buildings, instruments, and tools.

**Failure** The departure of software operation from its requirements or the nonperformance of an action due to the fault. Also the termination of the ability of an item to perform a required function or its inability to perform within previously specified limits.

**Failure mode and effects analysis (FMEA)** A procedure by which potential failure modes in a technical system are analyzed. A FMEA can be extended to perform what is called failure modes, effects, and criticality analysis (FMECA). In a FMECA, each failure mode identified is ranked according to the combined influence of its likelihood of occurrence and the severity of its consequences.

**Fault** An incorrect step, process, or data definition in a computer program. A fault is a defect found in executable software code. A fault is a subset of a defect.

**Fault isolation** The ability of a subsystem to prevent a fault within the subsystem from causing consequential faults in other subsystems.

**Fault-tree analysis** A systems engineering method for representing the logical combinations of various system states and possible causes that can contribute to a specified event (called the top event).

**Findings** The conclusions of an assessment, evaluation, audit, or review that identify the most important issues, problems, or opportunities within the area of investigation.

**Firmware** The combination of a hardware device and computer instructions or computer data that reside as read-only software on the hardware device. The software cannot be readily modified under program control.

**First-line manager** The first level of management responsible for reviewing performance and salary; the manager directly responsible for supervising the working-level staff. First-line managers are responsible for managing technical supervisors and professionals.

**First-line support** Also called first-level support or front-end support. The organizational unit that receives the initial customer call for support to his or her production software. Typically, a front-end or help-desk organization that can resolve high-level requests, logs the request in a management system, and reroutes the request when necessary.

**Formal method** A technique for expressing requirements in a manner that allows the requirements to be studied mathematically. Formal methods allow sets of requirements to be examined for completeness, consistency, and equivalency to another requirement set. Formal methods result in formal specifications.

**Formal procedure** A documented series of steps with guidelines for use.

**Formal review** A formal meeting at which a product is presented to the end user, customer, or other interested parties for comment and approval. It can also be a review of the management and technical activities and of the progress of the project.

**Formalization** Documentation, training, mentoring, deployment of improvements, and other mechanisms that seek to make a work process more formal, consistent, or standardized across the operating unit. (*See also* Institutionalization.)

**Frequency** A measure of likelihood expressed as the number of occurrences of an event in a given time. (*See also* Likelihood and Probability.)

**Full-time equivalent (FTE)** Representation of the size of a team of individuals based on their respective effort. As an example, three people have worked full time (5 days = 3 weeks) and one has worked 2 days (0.4 weeks), giving an FTE of 3.4.

**FSM method** A specific implementation of FSM defined by a set of rules, which conforms to the mandatory features of this part of ISO/IEC 14143.

**Function** In the programming sense, the implementation of an algorithm in the program with which the user or the program can perform all or part of a work task. In the functional analysis sense, an aspect of the intended behavior of the system. In the Human Resources (HR) sense, a set of related actions, undertaken by individuals or tools that are specifically assigned or fitted for their roles, to accomplish a set purpose or end (a task, action, or activity performed to achieve a desired outcome).

**Functional analysis** Examination of a defined function to identify all the subfunctions necessary to the accomplishment of that function, identification of functional relationships and interfaces (internal and external) and capturing these in a functional architecture, and flow down of upper-level performance requirements and assignment of these requirements to lower-level subfunctions.

**Functional architecture** The hierarchical arrangement of functions, their internal and external (external to the aggregation itself) functional interfaces and external physical interfaces, their respective functional and performance requirements, and design constraints.

**Functional baseline** The initially approved documentation describing a system's or configuration item's functional performance, interoperability, and interface requirements, and the verification required to demonstrate the achievement of those specified requirements.

**Functional domain** A class of software based on the characteristics of functional user requirements that are pertinent to FSM.

**Functional requirement** A requirement that defines a function of the system under development.

**Functional size** Size of the software derived by quantifying the functional user requirements.

**Functional user requirements** A subset of the user requirements. The functional

**user requirements** represent the user practices and procedures that the software must perform to fulfil the users' needs. They exclude quality requirements and any technical requirements.

**Generic practice** A practice that is added to the specific practices of a focus area to create a higher-capability process. A generic practice should be generally applicable to any focus area.

**Goals** A summary of the key practices of a key process area that can be used to determine whether an organization or project has effectively implemented the key process area. The goals signify the scope, boundaries, and intent of each key process area.

**Grade** Category or rank given to different quality requirements for products, processes, or systems having the same functional use.

**Group** The collection of departments, managers, and individuals who have responsibility for a set of tasks or activities. A group could vary from a single individual assigned part time, to several part-time individuals assigned from different departments, to several individuals dedicated full time.

**Guideline** Document stating recommendations or suggestions.

**Hazard** A source of potential harm or a situation with a potential to cause loss.

**Host computer** A computer used to develop software. (*See* Target computer for contrast.)

**Human/user centered** Approaches which have as their primary intention or focus the consideration of the interests or needs of the individuals and/or groups who will work with or use the output from a system.

**Impact analysis** The analysis activity associated with identifying the impact of a change to an existing software.

**Implementation** In the product-life-cycle context, the process of translating a product design into the product.

**Implementation model** A description of not only what, but how people, methods, materiel, and equipment are applied to produce an output or outcome.

**Implemented process** The process that members of programs in the organization actually do. Same as "performed process."

**Implied needs** Needs that may not have been stated but are actual needs when the entity is used in particular conditions.

**Improvement** Process perspective, *see* process improvement. Product perspective, an improvement to the existing software; synonymous with perfective maintenance.

**Incident** The name given to a defect during the testing cycle of a software. An incident has no impact on the customer as the software is not yet released at that time. Incidents are reported on incident reports.

**Indicator** A measure that can be used to estimate or predict another measure. *See* Assessment indicator.

**Indirect measure** A measure of an attribute that is derived from measures of one or more other attributes.

**Indirect measurement method** Measurement method designed from other measurement methods.

**Informal review** A review that is conducted in an ad-hoc manner.

**Information item** A defined group of elements of information. A subitem is a part of an information item.

**Infrastructure** All the systematic elements needed to sustain an initiative or effort. These are, at a minimum, people's skills and knowledge or the training to transition them into use, methods or techniques, materiel (inputs), facilities, and tools, which may or may not be automated.

**Initial level** See Maturity level.

**Initiating event** An event that can lead to a threat.

**Inspection** Activities such as measuring, examining, testing, gauging one or more characteristics of a product or service, and comparing these with specified requirements to determine conformity. The examination (review) of a product and its associated documentation to determine whether or not it conforms to requirements.

**Institutionalization** The building and reinforcement of infrastructure and corporate culture that support methods, practices, and procedures so that they are the ongoing way of doing business, even after those who originally defined them are gone.

**Instrumentation** The application of instruments (or measurements) for observation, measurement, or control.

**Integrated database** A database that contains work products and outcomes from implementation of the processes for engineering a system. Notes: 1. This database provides the information needed by the multidisciplinary teams and management to efficiently and effectively accomplish their assigned tasks. It typically contains the stakeholder needs, operational concept, operational requirements, and system requirements. 2. This database will contain (a) the current configuration of a system, (b) the current configuration baselines, and (c) all analysis and test results leading to decisions that affect the systems configuration or that are used in verifying the planned life-cycle events.

**Integrated product development (IPD)** A systematic approach to product (or service) development that achieves a timely collaboration of necessary disciplines throughout the product life cycle to better satisfy customer needs.

**Integrated product team (IPT)** A team that includes stakeholders in product success and holds them accountable for producing it, including later life-cycle disciplines. It may also include nonmember interfaces to less critical stakeholders, and affected groups that are not users (such as community representatives downwind of a chemical plant). Deploying IPTs does not necessarily constitute performing IPD.

**Integrated software management** The unification and integration of the software engineering and management activities into a coherent, defined software process based on the organization's standard software process and related process assets.

**Integration** The merger or combining of two or more elements (e.g., components, parts, or configuration items) into a functioning higher-level element with the functional and physical interfaces satisfied.

**Integrator** A supplier who takes the responsibility for all aspects of a deliverable. An integrator will typically be responsible for the deliverables of other suppliers.

**Integrity (of data)** Completeness and reliability of data stored in or processed by the software.

**Integrity assurance authority** The independent person or organization responsible for assessment of compliance with the integrity requirements.

**Integrity level** A denotation of a range of values of a property of an item necessary to maintain system risks within tolerable limits. For items that perform mitigating functions, the property is the reliability with which the item must perform the mitigating function. For items whose failure can lead to a threat, the property is the limit on the frequency of that failure.

**Interested party** Person or group having an interest in the performance or success of an organization.

**Interface partner** Individual or organization with whom arrangements for support of the software product are necessary in the operation, maintenance, and development processes.

**Intermediate software product** A product of the software development process that is used as input to another stage of the software development process.

**Internal attribute** A measurable property of an entity that can be derived purely in terms of the entity itself.

**Internal measure** A measure of the product itself, either direct or indirect.

**Internal quality** The totality of attributes of a product that determine its ability to satisfy stated and implied needs when used under specified conditions.

**Item** An entity such as a part, component, subsystem, equipment, or system that can be individually considered. An item may consist of hardware, software, or both.

**Key factors of functional user requirements** Critical, mandatory criteria expressed by the functional user requirements that have a significant or dominant influence on the software; for example, continuous operation is a key factor of avionics software.

**Key practices** The infrastructures and activities that contribute most to the effective implementation and institutionalization of a key process area.

**Key process area** A cluster of related activities in an area of software acquisition that, when performed collectively, achieve a set of goals considered important for establishing process capability in that area. The key process areas have been defined to reside at a single maturity level. These are the principal building blocks to help determine the software acquisition process capability of an organization and understand the improvements needed to advance to higher maturity levels.

**Life cycle** The scope of systems or product evolution beginning with the identification of a perceived customer need; addressing development, test, manufacturing, operation, support, and training activities; and continuing through various upgrades or evaluations until product disposal. The development, operation and maintenance of a system, spanning the life of the system from the definition of its requirements to the termination of its use. (*See also Software life cycle.*)

**Life-cycle model** A framework containing the processes, activities, and tasks involved in the development, operation, and maintenance of a software product, spanning the life of the system from the definition of its requirements to the termination of its use.

**Likelihood** Used as a qualitative description of probability and frequency.

**Local customization** A measurement method that has been modified for local use, such customization that it might produce different sizes than those obtained prior to modification.

**Loss** Any negative consequence, financial or otherwise.

**Maintainability plan** A document setting out the specific maintainability practices, resources, and sequence of activities relevant to software. The developer prepares the maintainability plan.

**Maintainer** An organization that performs maintenance activities.

**Maintenance** The process of modifying a product or component after delivery to correct faults, adapt to a changed environment, improve performance or other attributes, or perform line and depot maintenance of hardware components. Categories of software maintenance are user support, corrective, adaptive, preventive, or perfective.

**Maintenance enhancement** A maintenance enhancement is a software change that is not a software correction. There are two types of software enhancements: adaptive and perfective.

**Maintenance plan** A document setting out the specific maintenance practices, resources, and sequence of activities relevant to maintaining a software product. The maintainer prepares the maintenance plan. The plan should be activated once a product transitions to the maintenance phase.

**Maintenance process** The maintenance process contains the activities and tasks of the maintainer. This process is activated when the software product undergoes modifications to code and associated documentation due to a problem or the need for improvement adaptation. The objective is to modify an existing software product, preserving its integrity. This process includes the migration and retirement of the software product.

**Maintenance program** The organizational structure, responsibilities, procedures, processes, and resources used for implementing the maintenance plan. (The term “program” is synonymous with “infrastructure.”)

**Managed and controlled** The process of identifying and defining software work products that are not part of a baseline and, therefore, are not placed under configuration management, but that must be controlled for the project to proceed in

a disciplined manner. “Managed and controlled” implies that the version of the work product in use at a given time (past or present) is known (i.e., version control), and changes are incorporated in a controlled manner (i.e., change control).

**Management** Coordinated activities to direct and control an organization.

**Management practice** A management activity or task that addresses the implementation or institutionalization of a specific process attribute.

**Management system** System to establish policy and objectives and to achieve those objectives.

**Manager** A role that encompasses providing technical and administrative direction and control to individuals performing tasks or activities within the manager’s area of responsibility. The traditional functions of a manager include planning, resourcing, organizing, directing, and controlling work within an area of responsibility.

**Mandatory provision** Expression in the content of a normative document that takes the form of an instruction or a requirement and is denoted by the word “shall.”

**Marginal effectiveness** Effort is being expended but it is not clear that the benefit received for the effort invested is worth the cost of the effort. The effort could be removed without causing significant impact to the program or organization.

**Marginal value** Products are generated by the activity, but it is not clear that the products are of use to those for whom they are intended. The products could be removed without causing significant impact to the program or organization.

**Maturity level** A well-defined evolutionary plateau toward achieving a mature acquisition process.

**Maturity questionnaire** A set of questions about the software process that sample the key practices in each key process area of the CMM. The maturity questionnaire is used as a springboard to appraise the capability of an organization or project to execute a software process reliably.

**Measurably significant effectiveness** Effort being expended and the benefit are measured and found to be significant to the program or organization.

**Measurably significant value** The benefits of each product generated by the activity are measured and found to be of significant value to the program or organization.

**Measure (noun)** The number or category assigned to an attribute of an entity by making a measurement.

**Measure (verb)** Make a measurement.

**Measurement** Data collected on a process, task, or activity, or the action of collecting said data.

**Measurement process** Set of interrelated resources, activities, and influences related to a measurement

**Mechanism** A means or technique whereby the performance of a task, procedure, or process is assured. The mechanism may involve several organizational ele-

ments, and its documentation may include some combination of function statements, operating plans, position descriptions, and formal procedures. The documentation defines what should be performed, how it should be performed, and who is accountable for the results.

**Method** A reasonably complete set of rules and criteria that establishes a precise and repeatable way of performing a task and arriving at a desired result.

**Methodology** A collection of methods, procedures, and standards that defines an integrated synthesis of engineering approaches to the development of a product.

**Migration** Classified as a form of restructuring. Program migrations go from one language to another. Straight migrations will not alter the functionality of the software.

**Model** A simplified representation of some aspect of the real world.

**Modification request (MR)** A generic term used to identify proposed changes to a software product that is being maintained. The MR may later be classified as a correction or enhancement and categorized as corrective, adaptive, or perfective maintenance. MRs are also referred to as change requests.

**Monitor** To check, supervise, observe critically, or record the progress of an activity, action, or system on a regular basis in order to identify change.

**Monitoring** An examination of the status of a product or a process; for example, monitoring the activities of a supplier and of their results by the acquirer or a third party. Another example is monitoring the software in operation.

**Multidisciplinary teamwork** The cooperative application of all appropriate disciplines by people functioning as a team to achieve solutions that balance the contributions of the disciplines in an effective manner.

**Need** A user-related capability shortfall (such as those documented in a need statement, field deficiency report, or engineering-change order), or an opportunity to satisfy a new market or capability requirement because of a new technology application or breakthrough, or to reduce costs. Needs may also relate to providing a desired service (e.g., system disposal).

**Nonconformity** The nonfulfilment of specified requirements.

**Nondeliverable item** Hardware or software product that is not required to be delivered under the contract but may be employed in the development of a software product.

**Nondevelopmental item** An item of software that is available for delivery and acceptance prior to award of the contract.

**Nontechnical requirements** Agreements, conditions, and/or contractual terms that affect and determine the management activities of a software project.

**Normative** Relating to, or prescribing, a norm or standard.

**Objective evidence** Qualitative or quantitative information, records, or statements of fact pertaining to the characteristics of an item or service or to the existence and implementation of a process element, which is based on observation, measurement, or test, and which can be verified. (See also defined process.)

**Offered product** Product submitted to a customer external to the offering organization.

**Offeror** A contractor who submits a proposal in response to a solicitation package.

**Off-the-shelf product** Product that is already developed and available, usable either “as is” or with modification.

**Operational scenario** A sequence of events expected during operation of system products. Includes the environmental conditions and usage rates as well as expected stimuli (inputs) and responses (outputs).

**Operational software** The software that is intended to be used and operated in a system when it is delivered to the customer and deployed in its intended environment.

**Operational support** *See* User support.

**Operations product** One or more end products that, together, perform the operations function of a system.

**Operator** An individual or an organization who contributes to the functionality of a system and draws on knowledge and/or procedures to contribute the function.

**Opinion leader** An engineering professional highly respected by his/her peers and whose opinion can significantly influence his/her peers.

**Optimal effectiveness** Effort expended is providing maximum benefit for the amount of effort; that is, more effort results in a diminishing return to the program or organization.

**Optimal value** Value of the products generated by activity are of maximum utility to the program or organization.

**Optimizing level** *See* Maturity level.

**Optional** Information supplied at the discretion of the manufacturer or marketing organization.

**Optional provision** Expression in the content of a normative document that takes the form of a statement or a recommendation and is denoted by the word “should.”

**Organization** A company, firm, enterprise, or association, or other legal entity or part thereof, whether incorporated or not, public or private, that has its own function(s) and administration.

**Organization process maturity** The extent to which an organization has explicitly and consistently deployed processes that are documented, managed, measured, controlled, and continually improved. Organization process maturity may be measured via a process appraisal.

**Organizational process** A process described at the organizational level for use by programs in the organization. It may be a family of processes in order to capture the different classes of processes that frequently occur in organizations. It is intended that the organizational process be tailored into a well-defined process to meet the needs of specific programs.

**Organizational structure** Orderly arrangement of responsibilities, authorities, and relationships between people.

**Organizational unit** A single, defined organizational component (e.g., a department, section, project, or program).

**Organization's measurement program** The set of related elements for addressing an organization's measurement needs. It includes the definition of organization-wide measurements, methods, and practices for collecting organizational measurements and analyzing data, and measurement goals for the organization.

**Organization's software process assets** A collection of entities, maintained by an organization, for use by projects in developing, tailoring, maintaining, and implementing their software processes. These software process assets typically include: the organization's standard software process, descriptions of the software life cycles approved for use, the guidelines and criteria for tailoring the organization's standard software process, the organization's software process database, and a library of software-process-related documentation. Any entity that the organization considers useful in performing the activities of process definition and maintenance could be included as a process asset.

**Organization's software process database** A database established to collect and make available data on software processes and resulting software work products, particularly as they relate to the organization's standard software process. The database contains or references both the actual measurement data and the related information needed to understand the measurement data and assess it for reasonableness and applicability. Examples of process and work-product data include estimates of software size, effort, and cost; actual data on software size, effort, and cost; productivity data; peer review coverage and efficiency; and number and severity of defects found in the software code.

**Organization's standard software process** The operational definition of the basic process that guides the establishment of a common software process across the software projects in an organization. It describes the fundamental software process elements that each software project is expected to incorporate into its defined software process. It also describes the relationships (e.g., ordering and interfaces) between these software process elements.

**Orientation** An overview or introduction to a topic for those overseeing or interfacing with the individuals responsible for performing in the topic area. (See Train, for contrast.)

**Outsourcing** External provisioning of a good or service involving transferring the company's assets and personnel to the outsourcer as part of the agreement. Outsourcing contracts typically have 5–10 year terms.

**Package documentation** The product description and user documentation.

**Pareto analysis** The analysis of defects by ranking causes from most significant to least significant. Pareto analysis is based on the principle, named after the 19th-century economist Vilfredo Pareto, that most effects come from relatively few causes, i.e., 80% of the effects come from 20% of the possible causes.

**Partition** The assignment of logical arrangements of requirements to potential end products, manual operations, or associated processes.

**Peer review** A review of a work product, following defined procedures, by peers of the product's producer for the purpose of identifying defects and improvements.

**Peer review leader** An individual specifically trained and qualified to plan, organize, and lead a peer review.

**Perfective maintenance** The modification of a software product after delivery to improve performance or maintainability. Perfective maintenance provides enhancements necessary to improve software performance (e.g., person-machine interface enhancements), maintainability or other software attributes. Perfective maintenance does not implement new system requirements.

**Performance** A quantitative measure characterizing a physical or functional attribute relating to the execution of a mission or function. Performance attributes include quantity (how many or how much), quality (how well), coverage (how much area, how far), timeliness (how responsive, how frequent), and readiness (availability, mean time between failures). Performance is an attribute for all products, processes, and their associated personnel, including those for development, production, verification, deployment, operations, support, training, and disposal. Thus, supportability parameters, manufacturing process variability, reliability, and so forth, are performance measures.

**Performance criteria** Contract perspective: a contractual agreement that describes the performance objectives and contractual clauses in case of not meeting the performance. Application perspective: an application transaction performance (response-time) target. Process perspective: the turnaround time for conducting an activity.

**Performance requirement** How well the system products must perform a function, along with the conditions under which the function is performed.

**Performed process** What the members of the organization actually do. Also referred to as the "implemented process."

**Periodic review** A review that occurs at specified regular time intervals. (*See Event-driven basis, for contrast.*)

**Periodic review/activity** A review or activity that occurs at specified regular time intervals. (*See Event-driven review/activity, for contrast.*)

**Plan** A documented series of tasks required to meet an objective, typically including the associated schedule, budget, resources, organizational description, and work breakdown structure.

**Planning and control measure** A combination of measurements used to provide periodic assessment of the health and status of a program throughout its life cycle. These measures are used to detect the presence of adverse trends early enough so that corrective actions may be taken. Examples of planning and control measures include schedule performance versus plan, cost performance versus plan, staffing level actuals versus planned.

**Policy** A guiding principle designed to influence or determine decisions, actions, or other matters.

**Practice** A technical or management activity that contributes to the creation of the output of a process or enhances the capability of a process.

**Practitioner** Anyone who participates in the systems engineering process. Practitioners may be individuals responsible for accomplishing systems engineering tasks, individuals who use the products of systems engineering, or individuals who provide resources to enable systems engineering process implementation.

**Preventive action** Action taken to eliminate the cause of a potential nonconformity or other potentially undesirable situation.

**Prime contractor** An individual, partnership, corporation, or association that administers a subcontract to design, develop, and/or manufacture one or more products.

**Probability** The likelihood of a specific outcome, measured by the ratio of specific outcomes to the total number of possible outcomes. Probability is expressed as a number between 0 and 1, with 0 indicating an impossible outcome and 1 indicating an outcome is certain.

**Problem report** A term used to identify and describe failures detected in a production software product. Maintenance personnel will stop current work to attend to problems as priority work.

**Procedure** A written description of a course of action to be taken to perform a given task.

**Process** System of activities that uses resources to transform inputs into outputs. Also a sequence of steps performed for a given purpose; for example, the software development process.

**Process assessment** A disciplined evaluation of an organization's software processes against a model compatible with the reference model.

**Process asset library** A collection of process assets, maintained by an organization, for use by programs in developing, tailoring, maintaining, and implementing their defined processes. This collection exists within a defined architecture that gives structure to the example processes, process fragments, process-related documentation, process architectures, process tailoring rules and tools, and process measurements.

**Process assets** A collection of items, maintained by an organization, for use by programs in developing, tailoring, maintaining, and implementing their processes. These process assets typically include: the organization's standard processes, descriptions of the life-cycle models approved for use on programs, the guidelines and criteria for tailoring the organization's standard processes, the organization's measurement database, and a library of process-related documentation. Any item that the organization considers useful in performing the activities of process definition and maintenance could be included as a process asset.

**Process attribute** A measurable characteristic of process capability applicable to any process.

**Process attribute rating** A judgment of the level of achievement of the defined capability of the process attribute for the assessed process.

**Process capability** The range of expected results that can be achieved by following a process. The ability of a process to achieve a required goal. (*See* Process performance for contrast.)

**Process capability baseline** A documented characterization of the range of expected results that would normally be achieved by following a specific process under typical circumstances. A process capability baseline is typically established at an organizational level. (*See* Process performance baseline for contrast.)

**Process capability determination** A systematic assessment and analysis of selected software processes within an organization against a target capability, carried out with the aim of identifying the strengths, weaknesses, and risks associated with deploying the processes to meet a particular specified requirement.

**Process capability determination sponsor** The organization, part of an organization or person initiating a process capability determination.

**Process capability level** A point on the six-point ordinal scale (of process capability) that represents the increasing capability of the performed process; each level builds on the capability of the level below.

**Process capability level rating** A representation of the achieved process capability level derived from the process attribute ratings for an assessed process.

**Process category** A set of processes addressing the same general area of activity. In a maturity model, popular groupings are domains, key process areas, and road maps.

**Process context** The set of factors, documented in the assessment input, that influence the judgment, comprehension, and comparability of process attribute ratings.

**Process database** A repository into which all process data are entered. It is a centralized resource managed by the process group. Centralized control of this database ensures that the process data from all programs are permanently retained and protected.

**Process description** The operational definition of the major components of a process. Documentation that specifies, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a process. It may also include the procedures for determining whether these provisions have been satisfied. Process descriptions may be found at the task, project, or organizational level.

**Process development** The act of defining and describing a process. It may include planning, architecture, design, implementation, and validation.

**Process dimension** The set of processes comprising the functional aspects of the reference model of processes and process capability.

**Process domain** When used in the context of maturity models, a process domain is a regroupment of key process areas that relate logically to each other. For ex-

emple the process domain will include all key process areas that define, improve, measure, and innovate the processes of an organization.

**Process effectiveness** Process effectiveness is focused on the results of performing a process. These results are based on an organizational context that includes many attributes in addition to its process-related attributes. The aspects of effectiveness measured by the SECM are the approach to the process and its deployment. The approach is characterized by practices that are expected to produce increased benefit when performed appropriately in their organizational context. The deployment is characterized by goals that reflect the end state that performing a group of practices would be expected to exhibit.

**Process group** A group of specialists that facilitates the definition, maintenance, and improvement of the processes used by the organization.

**Process improvement** Action taken to change an organization's processes so that they meet the organization's business needs and achieve its business goals more effectively.

**Process improvement action** An action planned and executed to improve all or part of the software process.

**Process improvement program** All the strategies, policies, goals, responsibilities, and activities concerned with the achievement of specified improvement goals.

**Process improvement project** Any subset of the process improvement program that forms a coherent set of actions to achieve a specific improvement.

**Process management** The set of activities, methods, and tools applied to the definition, implementation, monitoring, and improvement of a process. Process management implies that a process is defined (since one cannot predict or control something that is undefined). The focus on process management implies that a program or organization takes into account both product and process-related factors in planning, performance, evaluation, monitoring, and corrective action.

**Process maturity** The extent to which a process is explicitly documented, managed, measured, controlled, and continually improved.

**Process measurement** The set of definitions, methods, and activities used to take measurements of a process and its resulting products for the purpose of characterizing and understanding the process.

**Process measures** Quantitative data used for assessing the effectiveness of the process and identifying corrective actions to be taken.

**Process outcome** An observable result of the successful implementation of a process.

**Process performance** The extent to which the execution of a process achieves its purpose. A measure of the actual results achieved by following a process. (See Process capability, for contrast.)

**Process performance baseline** A documented characterization of the actual results achieved by following a process. A process performance baseline is typically established at the project level, although the initial process performance

baseline will usually be derived from the process capability baseline. (See Process capability baseline, for contrast.)

**Process profile** The set of process attribute ratings for an assessed process.

**Process purpose** The high-level measurable objectives of performing the process and the likely outcomes of effective implementation of the process.

**Process tailoring** The activity of creating a process description by elaborating, adapting, and/or completing the details of process elements or other incomplete specifications of a process. Specific business needs for a project will usually be addressed during process tailoring.

**Producibility** A measure of the relative ease of manufacturing a product.

**Product** Any output or observable outcome of an activity or process, including those from services, intended for delivery to a customer or end user. This includes: (1) an item that is the goal of the engineering of a system, (2) a constituent part of the system, (3) goods and services. Note: An element performs one or more of the eight primary system functions: development, manufacturing, verification, deployment, operations, training, support, and disposal. The product will consist of one or more of the system element types: hardware, software, facilities, data, materiel, personnel, services, and techniques.

**Product description** A document stating properties of a software package, with the main purpose of helping potential buyers in the evaluation of the suitability for themselves of the product before purchasing it.

**Product liability** A generic term used to describe the onus on a producer or others to make restitution for loss related to personal injury, property, or other harm caused by a product or service. Also called service liability.

**Product measures** Measurable attributes of a product, such as size or number of defects, that generally do not vary over time (i.e., the product measure can be measured at any time).

**Production deviation permit** Written authorization, prior to production or before provision of a service, to depart from specified requirements for a specified quantity or for a specified time.

**Profile** A comparison, usually in graphical form, of plans or projections versus actuals, typically over time.

**Program** A set of tasks that are oriented toward meeting specific, defined objectives and accomplished by a group of individuals. The set of tasks are generally complex in nature and are performed within a definable time span (time between start and completion can often span numerous years) according to a planned schedule that has intermediate milestones.

**Program manager** The individual ultimately responsible for accomplishing the tasking and meeting the objectives, schedule, and fiscal constraints of the program. Everyone working on the program is tasked by the program manager, either directly or by delegated authority through subordinate managers.

**Project** Unique process, consisting of a set of coordinated and controlled activities with start and finish dates, undertaken to achieve an objective conforming to

specific requirements, including the constraints of time, cost, and resources. Also the structure of authorities, resources, and controls established by an organization to meet agreed upon requirements. Also an undertaking requiring concerted effort, which is focused on developing and/or maintaining a specific product. The product may include hardware, software, and other components. Typically, a project has its own funding, cost accounting, and delivery schedule.

**Project manager** The role with total business responsibility for an entire project; the individual who directs, controls, administers, and regulates a project acquiring software, a hardware/software system, or services. The project manager is the individual ultimately responsible to the end user.

**Project office** The aggregate of individuals assigned the primary responsibility for software acquisition in the contracted effort. A project office may vary in size from a single individual assigned part time to a large organization assigned full time.

**Project representatives** The engineering professionals representing a project to be assessed, typically the project leader and any optional support from one or two of the project technical professionals.

**Project team** All individuals that have an assigned software acquisition responsibility in the contracted effort. A project team may vary in size from a single individual assigned part time to a large organization assigned full time.

**Project's defined process** The operational definition of the process as used by a specific project. Well characterized and understood, it is described in terms of roles, standards, procedures, tools, and methods. It is developed by tailoring the organizational process to fit the specific characteristics and objectives of the project.

**Promotion** For SCI, the transition of state; for a baseline, the passage through the software life cycle.

**Proposed capability** The process capability that the organization proposes to bring to bear in meeting the specified requirement.

**Proposed change** A report of anomaly and the required or recommended enhancement from the time the idea is recorded until the disposition by the designated change authority.

**Prototype** A model (physical, electronic, digital, analytical, etc.) of a product built or constructed for the purpose of (a) assessing the feasibility of a new or unfamiliar technology, (b) assessing or mitigating technical risk, (c) validating requirements, (d) demonstrating critical features, (e) qualifying a product, (f) qualifying a process, (g) characterizing performance or product features, or (h) elucidating physical principles.

**Qualification** In the personnel sense, combination of personal attributes, minimum education, training, work, and audit experience, and competencies possessed by an auditor. In the V&V sense, the process of demonstrating whether an entity is capable of fulfilling specified requirements.

**Qualification requirement** A set of criteria or conditions that have to be met in

order to qualify a software product as complying with its specifications and being ready for use in its target environment.

**Qualification testing** Testing, conducted by the developer and witnessed by the acquirer (as appropriate), to demonstrate that a software product meets its specifications and is ready for use in its target environment.

**Qualify** (1) Determine formally that a design solution satisfies the specified requirements of the development baseline. (2) Declare suitability and readiness for production, deployment, training, operations, support, and disposition.

**Quality** Ability of a set of inherent characteristics of a product, system, or process to fulfil requirements of customers and other interested parties. Also the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs. Also the attribute(s) of a product, by which satisfaction of requirements is measured.

**Quality assurance** All those planned and systematic actions necessary to provide adequate confidence that a product or service will satisfy requirements for quality. Also, part of quality management focused on providing confidence that the relevant quality requirement will be fulfilled. (*See Software quality assurance.*)

**Quality audit** A systematic and independent examination to determine whether quality activities and related results comply with planned arrangements and whether these arrangements are implemented effectively and are suitable to achieve objectives.

**Quality characteristic** Inherent characteristic of a product, process, or system derived from a requirement.

**Quality control** The operational techniques and activities that are used to fulfil requirements for quality. Also part of quality management, focused on fulfilling quality requirements.

**Quality evaluation** Systematic examination of the extent to which an entity is capable of fulfilling specified requirements.

**Quality function deployment** A formal method for establishing and prioritizing customer needs and organizational capabilities to meet them, then tracing the deployment of processes and improved capabilities to deliver them through production and distribution of the product or service.

**Quality goals** Specific objectives that, if met, provide a level of confidence that the quality of a product is satisfactory.

**Quality improvement** Part of quality management, focused on increasing effectiveness and efficiency.

**Quality in use** The extent to which a product used by specified users meets their needs to achieve specified goals with effectiveness, productivity, and satisfaction in specified contexts of use.

**Quality loop** Conceptual model of interacting activities that influence the quality of a product or service in the various stages ranging from the identification of needs to the assessment of whether these needs have been satisfied. Also called quality spiral.

**Quality management** That aspect of the overall management function that determines and implements the quality policy. Also coordinated activities to direct and control an organization with regard to quality.

**Quality management system** System to establish a quality policy and quality objectives and to achieve those objectives.

**Quality manual** Document specifying the quality management system of an organization.

**Quality model** The set of characteristics and the relationships between them that provide the basis for specifying quality requirements and evaluating quality.

**Quality objective** Something sought, or aimed for, related to quality.

**Quality plan** Document setting out the specific quality practices, resources, and sequence of activities relevant to a particular product, service, contract, or request. Also document specifying the quality management system elements and the resources to be applied in a specific case.

**Quality planning** Part of quality management focused on setting quality objectives and specifying necessary operational processes and related resources to fulfil the quality objectives.

**Quality policy** Overall intentions and direction of an organization related to quality as formally expressed by top management.

**Quality requirement** Requirement for inherent characteristics of a product, process, or system. Any requirement relating to software quality as defined in a requirements quality model (for example, ISO 9126).

**Quality surveillance** The continuing monitoring and verification of the status of procedures, methods, conditions, processes, products, and services, and analysis of records in relation to stated preferences to ensure that specified requirements for quality are being met.

**Quality system** The organizational structure, responsibilities, procedures, processes, and resources for implementing quality management.

**Quality system review** A formal evaluation by top management of the status and adequacy of the quality system in relation to quality policy and new objectives resulting from changed circumstances.

**Quantitative control** Any quantitative or statistically based technique appropriate to analyze a software acquisition process, identify special causes of variations in the performance of the software acquisition process, and bring the performance of the software acquisition process within well-defined limits.

**Quantitative level** See Maturity level.

**Random error** A component of the error of measurement that, in the course of a number of measurement of the same attribute, varies in an unpredictable way.

**Rating** The action of mapping the measured value to the appropriate rating level. Used to determine the rating level associated with the software for a specific quality characteristic.

**Rating level** A scale point on an ordinal scale that is used to categorize a measurement scale.

**Record** Document stating results achieved or providing evidence of activities performed.

**Redocumentation** A form of software restructuring in which the resulting semantically equivalent representation is an alternate view intended for a human audience in a desired format.

**Reengineering** The examination and alteration of a subject system to reconstitute it in a new form, and the subsequent implementation of the new form. The new form is typically at the same or a higher level of abstraction as the original subject system.

**Refactoring** A behavior-preserving program transformation classified as restructuring, in the sense that it is always used in the context of object-oriented programming, and typically at the level of programs (i.e., source code). Restructuring is more general, in that it can be applied to any kind of software artifact, in any language, at any level of abstraction.

**Reference model** A model that is used as benchmark for measuring some attribute.

**Reference user requirements (RUR)** A set of user requirements used by a FSM method or reference FSM method to derive functional size results.

**Regrade** Alteration of the grade of a nonconforming product in order to make it conformant with requirements differing from the initial ones.

**Regression testing** The testing required to determine that a change to a system component has not adversely affected functionality, reliability, or performance.

**Rejuvenation** A general term (first used by Pfleeger) that includes all the techniques that aim at making the software more maintainable (reverse engineering, reengineering, restructureation, redocumentation, etc.)

**Release** A particular version of a configuration item that is made available for a specific purpose (for example, test release). Also an authorization to proceed to the next stage of a process.

**Reliability** The probability that a system or component will perform its required functions under stated conditions for a specified period of time. Also the ability of an item to perform a required function under stated conditions for a stated period of time.

**Repair** Action taken on a nonconforming product to make it acceptable for the intended usage.

**Repeatable level** See Maturity level.

**Repeatable process** A set of activities performed, in an essentially identical manner on many programs, to achieve a given purpose (process) that is: guided by organizational policies; documented and planned; allocated adequate resources (including funding, people, and tools); staffed with responsibilities assigned; implemented by trained individuals; measured; tracked with appropriate corrective actions; and reviewed by appropriate levels of management.

**Request for proposal** A document used by the acquirer as the means to announce

its intention to potential bidders to acquire a specified system, software product or software service. Also called tender.

**Request management system** The management system that captures the customer requests for service. Often a COTS system (Vantive, IncidentMonitor, c-Support, ServiceWise, Revelation, Ibehelppdesk, Heat, TrakIT, HelpTrack, Harmony, TechExcel, and many more) Often operated by the front-end support or help-desk organization. Typically, the software maintenance organization is a user of that system.

**Required training** Training designated by an organization to be required to perform a specific role.

**Requirement** Need or expectation that is stated, customarily implied or obligatory. Also something that governs that a product will have a given characteristic or achieve a given purpose, including what, how well, and under what conditions.

**Requirements analysis** The determination of system-specific performance and functional characteristics based on analyses of customer needs, requirements, and objectives; mission/operations; projected utilization environments for people, products, and processes; constraints; and measures of effectiveness. The bridge between customer requirements and system-specific requirements from which solutions can be generated for the primary system functions or characteristics.

**Requirements document** A document containing any combination of recommendations, requirements, or regulations to be met by a software package

**Requirements traceability** The evidence of an association between a requirement and its parent requirement or between a requirement and its implementation.

**Residual risk** The remaining level of risk after risk treatment measures have been taken.

**Resourcing, resourced** Having to do with identifying, acquiring, and applying or deploying resources needed by a process or activity.

**Responsibility** A duty to provide, or contribute in a particular way, a specified output or outcome, and the accountability to provide those expected results.

**Restructuring** A transformation from one form of representation to another at the same relative level of abstraction. The new representation is meant to preserve the semantics and external behavior of the original. If restructuring is applied to source code, it is a program transformation in the classical sense, but it could also be applied at higher levels of abstraction, such as designs or architectures.

**Retirement** Withdrawal of active support by the operation and maintenance organization, partial or total replacement by a new system, or installation of an upgraded system.

**Return on investment (ROI)** The ratio of earnings from output (product) to investment and production costs, which determines whether an organization benefits from performing an action to produce something.

**Reverse engineering** The process of analyzing a subject system with two goals in mind: (1) to identify the system's components and their interrelationships, and, (2) to create representations of the system in another form or at a higher level of abstraction. Reverse engineering is a process of examination only; the software system under consideration is not modified.

**Review** Activity undertaken to ensure the suitability, adequacy, effectiveness, and efficiency of the subject matter to achieve established objectives. (*See also* Formal review, Informal review, and Peer review.)

**Review data** The data that is gathered from requirements or design reviews.

These data are of two types. The first, concerning the review process, typically includes preparation time, errors identified during preparation (by category), hours per error found in preparation, review time, number of requirements or design statements reviewed, number of requirements or design statements re-reviewed per hour, and errors found per review man-hour (by category). The second type, product data from the review, typically includes errors found per requirement or design statement, action items identified from each review, action items closed for each review, items needing rereview, and rereviews conducted.

**Review efficiency** The percentage of errors found through the review process. It is typically stated as a percentage and is calculated by dividing the total errors found during review by the total errors found by both review and test through the completion of system integration test.

**Rework** Action taken on a nonconforming product to make it conform to the requirements.

**Risk** A function of the probability of occurrence of a given threat and the potential adverse consequences of that threat's occurrence. Also the uncertainty of achieving expectations. Also the possibility of suffering loss.

**Risk acceptance** An informed decision to accept the likelihood and the consequences of a particular risk.

**Risk analysis** Systematic use of available information to determine how often specified events may occur and the magnitude of their likely consequences.

**Risk assessment** The overall process of risk analysis and risk evaluation.

**Risk avoidance** An informed decision not to become involved in a risk situation.

**Risk control** The part of risk management that involves the provision of policies, standards, and procedures to eliminate, avoid, or minimize adverse risks facing an enterprise.

**Risk identification** The process of determining what can happen, why, and how.

**Risk management** The systematic application of management policies, procedures, and practices to the tasks of identifying, analyzing, evaluating, treating, and monitoring risk.

**Role** A unit of defined responsibilities that may be assumed by one or more individuals.

**Safety** The expectation that a system does not, under defined conditions, lead to a state in which human life, health, property, or the environment is endangered.

**SCI types** The many types of electronic documents, files, diagrams, models, schema, and so on administered by SCM.

**Scrap** Action taken on a nonconforming product to preclude its originally intended usage.

**Second-level support** Second-level support, in organizations that have a first-level support, concerns all the software maintenance services offered to the customer. Typical second-level supports provided by software maintenance and often requested by users are: (a) general help on software to users; (b) one-time data extracts and ad-hoc reports to users, developers, and operations; (c) corrective work (failure repairs); (d) adaptive work; (e) perfective work; (f) preventive work; and (g) data recovery.

**Security** The protection of information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them.

**Self-appraisal** A systems-engineering appraisal for which (a) all or most of the appraisal team is from the appraised organization (not necessarily the appraised site) and (b) the appraised organization has primary responsibility for facilitation and planning.

**Senior manager** A management role at a high enough level in an organization that the primary focus is the long-term vitality of the organization, rather than short-term project and contractual concerns and pressures. In general, a senior manager for engineering would have responsibility for multiple projects.

**Sensitivity analysis** Examines how the results of a calculation or model vary as individual assumptions are changed.

**Service** An intangible product that is the result of at least one activity performed at the interface between the supplier of software maintenance and the customer.

**Service call** A customer request to the software maintenance personnel for software maintenance service. Can be direct or through a help desk or another front-end support organization.

**Service level agreement (SLA)** An approved document between software maintenance and customers (sometimes contractual) in which the services are defined, performance level targets documented, and budgets for software documented.

**Significant effectiveness** Effort being expended is obviously beneficial to the program or organization.

**Significant value** Products generated by the activity are obviously beneficial to those who use them. Products of significant value are avidly sought out and used by those for whom they are intended.

**Simulation** A model of a product or its component(s) in operation. Simulations may be computer-based analogies to a product such as video games, or a practice

role-play of an operation with or without tools (a simulation of a method or technique as in training classes).

**Small enhancement** Synonym of minor enhancement and small change. Refactoring or change that requires little effort from the maintenance personnel. Change means the design and development of minor functionality or data enhancement on existing software. Includes small (meaning less than 5 days of effort) changes to documentation, interfaces, source code, or data structure of existing software. Applies to (a) adaptive, (b) perfective, and (c) preventive modification requests.

**Software** All or part of the programs, procedures, rules, and associated documentation of an information processing system. Also an intellectual product consisting of information on a support medium. Intellectual creation comprising the programs, rules, and associated data that, when loaded into the program execution area of a computer, enables that computer to operate.

**Software acquisition management personnel** Those individuals who are trained, educated, or experienced in software acquisition management and who are either assigned to or support the project team in the performance of software acquisition activities.

**Software acquisition plans** The collection of plans, both formal and informal, used to express how software acquisition activities will be performed; for example, software acquisition risk management plan or project management plan.

**Software acquisition process** A set of activities, methods, practices, and transformations that people use to acquire software and associated products.

**Software acquisition process assets** A collection of entities, maintained by an organization, for use by projects in developing, tailoring, maintaining, and implementing their software acquisition processes. Some examples of these software acquisition process assets include: the acquisition organization's standard software acquisition process, descriptions of the software life cycles approved for use, the guidelines and criteria for tailoring the acquisition organization's standard software acquisition process, the organization's software acquisition process database, and a library of software acquisition process-related documentation. Any entity that the organization considers useful in performing the activities of process definition and maintenance could be included as a process asset.

**Software acquisition process group** This group is responsible for the definition, improvement, and maintenance of the acquisition organization's standard software acquisition process and related process assets, including guidelines for all projects to tailor the standard software acquisition process to their specific situations. It coordinates process activities with the software projects and related elements of the organization.

**Software acquisition process repository** A collection of software acquisition process information (e.g., estimated and actual data on software project size, effort, and cost; and project team productivity and quality data) gathered from the software acquisition projects and maintained by the acquisition organization to support its software acquisition definition and improvement activities.

**Software acquisition process-related documentation** Documents and document fragments that may be of use to future project teams when tailoring the acquisition organization's standard software acquisition process. Examples may cover subjects such as a project's defined software acquisition process, standards, procedures, software acquisition risk management plans, and training materials.

**Software acquisition project** An undertaking that is focused on acquiring the software components and associated documentation of a system. A software project may be part of a project building a hardware or software system.

**Software-acquisition-related group** A collection of individuals (both managers and technical staff) representing a software discipline that supports, but is not directly responsible for, software acquisition. Examples of software disciplines include software configuration management and software quality assurance.

**Software architecture** The organizational structure of the software or module.

**Software baseline audit** An examination of the structure, contents, and facilities of the software baseline library undertaken to verify that baselines conform to the documentation that describes the baselines.

**Software baseline library** The contents of a repository for storing configuration items and associated records.

**Software build** An operational version of a software system or component that incorporates a specified subset of the capabilities that the final software system or component will provide.

**Software capability evaluation** An appraisal by a trained team of professionals to identify contractors who are qualified to perform the software work or to monitor the state of the software process used on an existing software effort.

**Software configuration control board** A group responsible for evaluating and approving or disapproving proposed changes to configuration items, and for ensuring implementation of approved changes.

**Software configuration item (SCI)** Called "configuration item" in ISO 10007, excluding hardware and services. Used recursively to identify individual software items of significance.

**Software configuration management (SCM)** The process of applying configuration management (*see ISO 10007*) throughout the software life cycle to ensure the completeness and correctness of SCIs.

**Software configuration management plan (SCM plan)** Planning information about SCM activities. Called "configuration management plan" in ISO 10007.

**Software development plan** The collection of plans that describe the activities to be performed for the software project. It governs the management of the activities performed by the software engineering group for a software project. It is not limited to the scope of any particular planning standard.

**Software engineering environment (SEE)** The set of automated tools, firmware devices, and hardware necessary to perform the software engineering effort. The automated tools may include but are not limited to compilers, assemblers, link-

ers, loaders, operating systems, debuggers, simulators, emulators, test tools, documentation tools, and database management systems.

**Software engineering group** The collection of individuals (both managers and technical staff) who have responsibility for software development and maintenance activities (i.e., requirements analysis, design, code, and test) for a project. Groups performing software-related work, such as the software quality assurance group, the software configuration management group, and the software engineering process group, are not included in the software engineering group.

**Software engineering personnel** Those individuals who are trained, educated, or experienced in software engineering and who are either assigned to or support the project team in the performance of software acquisition activities.

**Software engineering process group** A group of specialists who facilitate the definition, maintenance, and improvement of the software process used by the organization. In the key practices, this group is generically referred to as “the group responsible for the organization’s software process activities.”

**Software engineering staff** The software technical people (e.g., analysts, programmers, and engineers), including software task leaders, who perform the software development and maintenance activities for the project, but who are not managers.

**Software integration** A process of putting together selected software components to provide the set or specified subset of the capabilities the final software system will provide.

**Software integrity level** The integrity level of a software item.

**Software library** A controlled collection of SCIs designated to keep those with like status and type together and segregated from unlike, to aid in development, operation, and maintenance.

**Software life cycle** The period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and, sometimes, retirement phase.

**Software manager** Any manager, at a project or organizational level, who has direct responsibility for software development and/or maintenance.

**Software plans** The collection of plans, both formal and informal, used to express how software development and/or maintenance activities will be performed. Examples of plans that could be included in this category are software development plan, software quality assurance plan, software configuration management plan, software test plan, risk management plan, and process improvement plan.

**Software process** A set of activities, methods, practices, and transformations that people use to develop and maintain software and associated products (e.g., project plans, design documents, code, test cases, and user manuals).

**Software process assessment** An appraisal by a trained team of software profes-

sionals used to determine the state of an organization's current software process, to determine the high-priority, software-process-related issues facing an organization, and to obtain the organizational support for software process improvement.

**Software process description** The operational definition of a major software process component identified in the project's defined software process or the organization's standard software process. It documents, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a software process. (*See also* Process description.)

**Software process element** A constituent element of a software process description. Each process element covers a well-defined, bounded, closely related set of tasks (e.g., software estimating element, software design element, coding element, and peer review element). The descriptions of the process elements may be templates to be filled in, fragments to be completed, abstractions to be refined, or complete descriptions to be modified or used unmodified.

**Software process improvement plan** A plan, derived from the recommendations of a software process assessment, that identifies the specific actions that will be taken to improve the software process and outlines the plans for implementing those actions. Sometimes referred to as an action plan.

**Software process improvement proposal** A documented suggestion for change to a process or process-related item that will improve software process capability and performance. (*See also* Action proposal.)

**Software process maturity** The extent to which a specific process is explicitly defined, managed, measured, controlled, and effective. Maturity implies a potential for growth in capability and indicates both the richness of an organization's software process and the consistency with which it is applied in projects throughout the organization.

**Software process-related documentation** Example documents and document fragments, which are expected to be of use to future projects when they are tailoring the organization's standard software process. The examples may cover subjects such as a project's defined software process, standards, procedures, software development plans, measurement plans, and process training materials.

**Software product** The complete set, or any of the individual items of the set, of computer programs, procedures, and associated documentation and data designated for delivery to a customer or end user (IEEE STD 610). (*See* Software work product for contrast.)

**Software product developer** The person or organization that manufactures a software product.

**Software product evaluation** Technical operation that consists of producing an assessment of one or more characteristics of a software product according to a specified procedure.

**Software project** An undertaking requiring concerted effort, which is focused on analyzing, specifying, designing, developing, testing, and/or maintaining the

software components and associated documentation of a system. A software project may be part of a project building a hardware or software system.

**Software quality assurance** (1) A planned and systematic pattern of all actions necessary to provide adequate confidence that a software work product conforms to established technical requirements. (2) A set of activities designed to evaluate the process by which software work products are developed and/or maintained.

**Software quality goal** Quantitative quality objectives defined for a software work product.

**Software quality management** The process of defining quality goals for a software product, establishing plans to achieve these goals, and monitoring and adjusting the software plans, software work products, activities, and quality goals to satisfy the needs and desires of the customer and end users.

**Software-related contractual requirements** All technical and nontechnical requirements related to the software portion of the acquisition.

**Software-related group** A collection of individuals (both managers and technical staff) representing a software engineering discipline that supports, but is not directly responsible for, software development and/or maintenance. Examples of software engineering disciplines include software quality assurance and software configuration management.

**Software requirement** A condition or capability that must be met by software needed by a user to solve a problem or achieve an objective (IEEE STD 610).

**Software service** Performance of activities, work, or duties connected with a software product, such as its development, maintenance, and operation.

**Software support** A term describing the overall activities of the software maintenance personnel.

**Software test environment (STE)** The facilities, hardware, software, firmware, procedures, and documentation needed to perform qualification and possibly other testing of software. Elements may include but are not limited to simulators, code analyzers, test-case generators, and path analyzers, and may also include elements used in the software engineering environment.

**Software tool** A software product providing automatic support for software life-cycle tasks.

**Software transition** A controlled and coordinated sequence of actions wherein software development passes from the organization performing initial software development to the organization performing software maintenance. Transition planning starts early during the development of new software. Maintenance personnel are involved at many stages of the development to ensure that the final transfer from development to maintenance is made easy.

**Software unit** A separately compilable piece of code.

**Software work product** Any artifact created in the process part of defining, maintaining, or using a software process, including process descriptions, plans, procedures, computer programs, and associated documentation, which may or

may not be intended for delivery to a customer or end user. (See Software product for contrast.)

**Solicitation package** When seeking suppliers for a particular acquisition, it is the distributed information that tells the interested bidders what the requirements are, how to prepare their proposals, how proposals will be evaluated, and when to submit their proposals. Sometimes called request for proposals (RFP).

**Special cause (of a defect)** A cause of a defect that is specific to some transient circumstance and not an inherent part of a process. Special causes provide random variation (noise) in process performance. (See Common cause for contrast.)

**Special causes of variation** Special causes of variation are assignable to people, places, materials, events, and so forth. They are causes of variation that are not attributable to, or inherent in, the process itself, although they may be attributable to some aspect of its execution.

**Specific practice** A practice contained in a focus area that describes an essential activity to accomplish the purpose of the focus area, is not generic to all Focus Areas, and contains a description, typical work products in which one might find evidence that the practice is performed and how well it is performed, and notes, including explanations, context, references for “how to,” and so on.

**Specification** A document that clearly and accurately describes requirements and other characteristics of a product and the procedures to be used to determine that the product satisfies these requirements.

**Sponsor** The senior site executive who has committed to or requested the systems engineering appraisal. This person typically has control of financial and other resources for the systems engineering organization.

**Staff** The individuals, including task leaders, who are responsible for accomplishing an assigned function, such as software development or software configuration management, but who are not managers.

**Stage** The high-level life-cycle classification used to facilitate management of a system.

**Staged architecture** A capability model in which sets of focus areas are defined as maturity levels that must be completed before other sets are attempted. This explicit prioritization of focus areas establishes an infrastructure or other dependencies perceived to exist by the model developers or community. Contrast with the continuous architecture, in which all focus areas have explicit equal priority and increasing process capability is within those focus areas chosen by the organization to improve.

**Stakeholder** An individual or organization interested in the success of a product or system. Examples of stakeholders include customers, developers, engineering, management, manufacturing, and users. Those people and organizations who may affect, be affected by, or perceive themselves to be affected by, the decision or activity.

**Stakeholder requirement** Represents the needs and expectations of a stakeholder.

**Standard** A document that establishes engineering and technical requirements for processes, procedures, practices, and methods that have been decreed by authority or adopted by consensus.

**Standard process** The operational definition of the basic process that guides the establishment of a common process in an organization.

**Standard software acquisition process** *See* Software acquisition process.

**State** A status indicator.

**Statistical quality control** Statistically based analysis of a process; measurements of process performance, including identification and elimination of common and special causes of variations in the process; and the designation and implementation of improvements to change the process performance in the direction of improved customer value or operational efficiency.

**Subcontractor** An individual, partnership, organization, enterprise, or association that contracts with an organization to perform, or help to perform, some part(s) of the contracting organization's product life cycle: management, requirements, design, development, production, delivery, support, and/or disposal of a product or its by-products.

**Submitter** The person or organization submitting the candidate FSM method and its documentation for compliance assessment.

**Subsystem** A grouping of items that will perform a logical set of functions within a particular end product.

**Supplier** An organization that enters into a contract with the acquirer for the supply of a system, software product, or software service under the terms of the contract.

**Supporting function** An organization responsible for assisting the software evaluation activities through the provision of technology, tools, experiences, and management skills.

**Sustainability** The product attribute of being maintainable in an operable condition, whether the product is in use or not, by customers or end users. Includes supply aspects if the product consumes materiel.

**Synthesis** The translation of input requirements (including performance, function, and interface) into possible solutions (resources and techniques) satisfying those inputs. Defines a physical architecture of people, product, and process solutions for a logical grouping of requirements (performance, function, and interface) and then designs those solutions.

**System** An object consisting of interrelated or interacting elements. A collection of components organized to accomplish a specific function or set of functions. Also an object consisting of interrelated or interacting elements

**System architecture** A logical, physical structure that specifies interfaces and services provided by the system components necessary to accomplish system functionality.

**System component** A basic part of a system. System components may be personnel, hardware, software, facilities, data, materiel, services, and/or techniques that

satisfy one or more requirements in the lowest levels of the functional architecture. System components may be subsystems and/or configuration items.

**System design process** A process for converting stakeholder requirements into design solutions.

**System engineering group** The collection of individuals (both managers and technical staff) who have responsibility for specifying the system requirements; allocating the system requirements to the hardware, software, and other components; specifying the interfaces between the hardware, software, and other components; and monitoring the design and development of these components to ensure conformance with their specifications.

**System integrity level** The integrity level of a system.

**System requirement** A condition or capability that must be met or possessed by a system or system component to satisfy a condition or capability needed by a user to solve a problem.

**System requirements allocated to software requirements** The subset of the system requirements that are to be implemented in the software components of the system. The allocated requirements are a primary input to the software development plan. Software requirements analysis elaborates and refines the allocated requirements and results in software requirements that documented.

**System technical requirement** A deconflicted set of stakeholder requirements stated in technical terms.

**Systematic error** A component of the error of measurement that remains constant, or varies in a predictable way, over the course of a number of measurements of the same object.

**Systematic failure** A failure related in a deterministic way to a certain cause, which can only be eliminated by a modification of the design or of the manufacturing process, operational procedures, documentation, or other relevant factors.

**Systems engineering** An interdisciplinary approach and means to enable the realization of successful systems.

**Systems engineering process group (SEPG)** A group focused on improving the systems engineering process used by an organization. The SEPG defines and documents the systems engineering process, establishes and defines process metrics, supports program data gathering, assists programs in analyzing data, and advises management of areas requiring further attention.

**Tailor** To modify a process, standard, or procedure to better match process or product requirements.

**Tailored process** A defined process developed by tailoring a standard process definition.

**Tailoring** The process by which the requirements of specifications, standards, and related documents are modified to make them suitable for a specific application or program.

**Target capability** The process capability that the process capability determina-

tion sponsor judges will represent an acceptable process risk to the successful implementation of the specified requirement.

**Target computer** The computer on which delivered software is intended to operate. (See Host computer for contrast.)

**Task** Well-defined unit of work in a process that provides a visible checkpoint of the status of the product of the process. Tasks have readiness (entry) criteria and completion (exit) criteria. Activities are informal tasks or steps within a task.

**Task kick-off meeting** A meeting held at the beginning of a task of a project for the purpose of preparing the individuals involved to perform the activities of that task effectively.

**Task leader** The leader of a technical team for a specific task, who has technical responsibility and provides technical direction to the staff working on the task.

**Team** A group of individuals who are mutually responsible for achieving goals. Teams usually are limited to five to nine members, but may include interface representatives for stakeholders in the team's goals, and may appear much larger, as in teams of team representatives, for example.

**Technical objectives** Technical objectives or goals guide the development effort by providing "target" values for item characteristics. These can include cost, schedule, and performance attributes deemed important. Technical objectives are not specification requirements.

**Technical performance measurement (TPM)** The technique of predicting the future value of key technical parameters of the end system based on current assessments of systems that make up that end system. Notes:

1. Involves the continuing verification of the degree of anticipated and actual achievement for technical parameters. Confirms progress and identifies variances that might jeopardize meeting an end-system requirement. Assessed values falling outside established tolerances indicate a need for evaluation and corrective action.
2. Key characteristics of TPM are
  - (a) *Achievement-to-date* presents achieved value of the technical parameter based on estimates or actual measurement.
  - (b) *Current estimate* is the value of the technical parameter predicted to be achieved by the end of the technical effort with remaining resources (including schedule and budget).
  - (c) *Technical milestone* is a point at which TPM evaluation is accomplished or reported.
  - (d) *Planned value profile* is the time-phased achievement projected for the technical parameter from the beginning of the development or as re-planned as a result of a corrective projection.
  - (e) *Tolerance band* is an envelope containing the planned value profile and indicating the allowed variation and projected estimation error.

- (f) *Objective* is the goal or desired value at the end of the technical effort.
- (g) *Threshold* is the limiting acceptable value that, if not met, would jeopardize the program.
- (h) *Variation* is the difference between the planned value and the achievement-to-date value.

**Technical requirements** Requirements relating to the technology and environment, for the development, maintenance, support, and execution of the software.

**Technical software requirements** The system requirements allocated to software.

**Techniques** Methods and skills required to carry out a specific task.

**Technology** The application of science and/or engineering to accomplish a particular result.

**Test** An activity in which a system, product, or a component is used under specified conditions, the results are observed or recorded, and an evaluation is made as to whether it adequately meets some or all of its requirements. Also a technical operation that consists of the determination of one or more characteristic of a given product, process, or service according to a specified procedure.

**Test case** A documented instruction for the tester that specifies how a function or a combination of functions shall or should be tested.

**Test coverage** The extent to which the test cases test the requirements for the system or software product.

**Testability** (1) The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met. (2) The degree to which a requirement is stated in terms that permit establishment of test criteria and performance of tests to determine whether those criteria have been met.

**Third-level support** In this book, it represents the technical support organizations located in the computer operations (workstation hardware and software, cabling, telecommunications, central servers, base software, and database). Operations may have to turn to vendors for specific support.

**Threat** A state of the system or system environment that can lead to adverse effects in one or more given risk dimensions.

**Ticket** Record of a software maintenance request that can be routed to the personnel. Can be of the following types: (a) user support requests, (b) modification requests (MRs), and (c) problem reports (PRs). Tickets are created in a request management system often shared or owned by other IT organizations (help desk, front-end support, and operations). Tickets are often constrained by the service level agreement rules for service. A specific entry has a unique number often called the ticket number, which is the number of the customer request or problem.

**Tool** Typically, a computer program used to help automate tasks associated with the definition, synthesis, test, analysis, or maintenance of models, designs, and documentation associated with systems components.

**Top management** Person or group of people who direct and control an organization at the highest level.

**Total quality management (TQM)** An industry term for a variety of improvement methods and techniques that generally focus on process improvement using measurement.

**Traceability** The ability to trace, in both the forward and backward directions, the lineage of a requirement from its first-level inception and subsequent refinement to its implementation in a software product, and the documentation associated with the software product.

**Trade study** An objective evaluation of alternative requirements, architectures, design approaches, or solutions using identical ground rules and criteria.

**Trade-off** Decision-making actions that select from various requirements and alternative solutions on the basis of net benefit to the stakeholders.

**Train** To make proficient with specialized instruction and practice. (*See also Orientation, for contrast.*)

**Training** Instruction and applied exercises for the attainment and retention of skills and knowledge required to accomplish necessary tasks.

**Training group** The collection of individuals (both managers and staff) who are responsible for coordinating and arranging the training activities for an organization. This group typically prepares and conducts most of the training courses and coordinates use of other training vehicles.

**Training program** The set of related elements that focuses on addressing an organization's training needs. It includes an organization's training plan, training materials, development of training, conduct of training, training facilities, evaluation of training, and maintenance of training records.

**Training waiver** A written approval exempting an individual from training that has been designated as required for a specific role. The exemption is granted because it has been objectively determined that the individual already possesses the needed skills to perform the role.

**Transition** The process of transferring responsibility for the acquired software products from the project manager to the software support organization.

**Transitioned** Indicates that a tool, technique, method, process, or product has been placed into use in an operational environment where it was not used before. The implication is that personnel or users were trained, and the method or product is actually the way that they do work now or the object that they use now.

**Unit** (1) A separately testable element specified in the design of a computer software component. (2) A logically separable part of a computer program. (3) A software component that is not subdivided into other components.

**User** An individual or organization that uses the operational system to perform a specific function. Also any person who specifies functional user requirements and/or any person or thing that communicates or interacts with the software at

any time. Also an individual or group who is the intended beneficiary of system use. (*See also End user.*) Anyone who employs an artifact or system to achieve a task.

**User documentation** The complete set of documents, available in printed or non-printed form, that is provided for the application of the product and also is an integral part of the product.

**User requirements baseline** A configuration baseline of user requirements. This baseline is necessary to systems engineering domains in which there does not exist a single, clearly discernible user or customer.

**User support** Synonym of operational support. One of the software maintenance services. Consists of helping end users understand the software functionality, documentation, data, and business rules. A support activity implies that no permanent change is done to the production software. Development of ad-hoc queries, reports, and extraction of data, as one time activities, is considered support work by maintenance personnel.

**Validation** The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements (IEEE STD 610).

**Value** A measure of the desirability of the products of an activity. SECM characterizes value as marginal, adequate, significant, measurably significant, and optimal.

**Variation** Difference between the planned value of the technical parameter and the achievement-to-date value derived from analysis, test, or demonstration.

**Verification** The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase (IEEE STD 610). An activity that ensures that the selected design solution satisfies the detailed technical requirements.

**Verify** To prove to be true by demonstration, evidence, or testimony.

**Version** An identified instance of an SCI within a schema for the purpose of revealing change history. The SCI with a specific identification released to software engineers for drafting, review, or modification, or to the acquirer for use.

**Waiver** A document stating a cancellation or reduction of a requirement.

**Well-defined process** A documented, consistent, and complete process with entry criteria, inputs, task descriptions, verification descriptions and criteria, outputs, and exit criteria.

**Work breakdown structure (WBS)** A product-oriented family tree composed of hardware, software, services, data, and facilities that results from systems engineering efforts and which completely defines the program. Displays and defines the product(s) to be developed or produced, and relates the elements of work to be accomplished to each other and to the end product.

**Work environment** Set of conditions under which a person operates.

**Work product** Anything produced by a process. This includes files, documents, components, work in progress, specifications, invoices, and so forth, generated during process performance, not just the product delivered to the process customer or user.

**Written procedure** *See* Procedure.

# References

Note: Date of Internet access is in brackets.

- Abran, A., Nguyenkim, H., 1991: Analysis of Maintenance Work Categories Through Measurement. In *Proceedings of the IEEE Conference on Software Maintenance*, Sorrento, Italy, October, pp. 104–113.
- Abran, A., Nguyenkim, H., 1993a: Measurement of the Maintenance Process from a Demand-Based Perspective. *Journal of Software Maintenance: Research and Practice*, 5, 2, 63–90.
- Abran, A., 1993b: Maintenance Productivity and Quality Studies: Industry Feedback on Benchmarking. In *Proceedings of the Software Maintenance Conference (ICSM93)*, Montréal, September.
- Abran, A., Maya, M., 1995: A Sizing Measure for Adaptive Maintenance Work Products. In *ICSM-95*, Opio, France, October 1995, pp. 286–294.
- Abran, A., Moore, J.W., Bourque, P., Dupuis, R., 2005: *Guide to the Software Engineering Body of Knowledge (SWEBOK)*, IEEE Computer Society Press, Los Alamitos, CA. <http://www.swebok.org> [25 April 2007].
- April, A., 2002a: Revue Critique de la Littérature: La Maintenance du Logiciel, Internal Technical Report 02-001, Montréal, ÉTS Software Engineering Laboratory, 10-11-2002.
- April, A., 2002b: Revue Critique de la Littérature: Partie 2, Les Modèles de Référence Pour l'évolution des processus du logiciel, Internal Technical Report 02-002, Montréal, ÉTS Software Engineering Laboratory, 30-12-2002.
- April, A., Al-Shuroogi, D., 2000: Software Product Measurement for Supplier Evaluation. In *Proceedings of the Software Measurement Conference (FESMA-AEMES)*, Madrid, Spain, October 18–20, 2000. <http://www.lrgl.uqam.ca/publications/pdf/583.pdf> [25 April 2007].
- April, A., Bouman, J., Abran, A., Al-Shuroogi, D., 2001: Software Maintenance in an SLA: Controlling the Customer Expectations. In *Proceedings of the Fourth European Software Measurement Conference (FESMA2001)*, Heidleberg, Germany, pp. 39–47.
- April, A., Abran, A., Dumke, R., 2004a: SM<sup>cmm</sup> to Evaluate and Improve the Quality of Software Maintenance Process: Overview of the Model. In *Proceedings of the Conference on Process Assessment and Improvement, Critical Software SA (SPICE2004)*. Lisbon, April, pp. 19–32.
- April, A., Abran, A., Dumke, R., 2004b: Assessment of Software Maintenance Capability: A Model and its Architecture. In *Proceedings of the 8th European Conference on Software Maintenance and Reengineering (CSMR2004)*. Tampere, Finland, March, pp. 243–248.
- April, A., Abran, A., Dumke, R., 2004c: Software Maintenance Productivity Measurement:

- How to Assess the Readiness of Your Organisation. In *Proceedings of the International Conference on Software Measurement (IWSM/MetriKon)*, Königs Wusterhausen, Germany, pp. 231–244.
- April, A., Abran, A., Dumke, R., 2004d: Assessment of Software Maintenance Capability : A Model and its Design Process, Conference on Software Engineering, Innsbruck Austria, February.
- April, A., Dumke, R., Abran, A., 2004e: SM<sup>mm</sup> Model to Evaluate and Improve the Quality of the Software Maintenance Process, Preprint Nr. 13, Facultät für Informatik, University of Magdeburg, November.
- April, A., Abran, A., Dumke, R., 2004f: What You Need to Know about Software Maintenance, *Maintenance Journal*, 17, 1, pp. 10–13.
- April, A., Coallier, F., 1995a: TRILLIUM V3.0: A Model for the Assessment of Telecom Software Development Capability. In *Proceedings 2nd International SPICE Symposium*, edited by T.P. Rout, AQRI, Brisbane, Australia, June, pp. 79–88.
- April, A., Coallier, F., 1995b: TRILLIUM: A Model for the assessment of Telecom Software System Development and Maintenance Capability. In *Proceedings of the IEEE International Software Symposium*, IEEE Computer Society Press, Los Alamitos, CA, pp. 175–183.
- Arnold, R.S., Bohner, S., 1996: *Software Change Impact Analysis*, Wiley–IEEE Press, New York.
- Arnold, R.S., 1994: *Software Reengineering*, IEEE Computer Society Press, Los Alamitos, CA.
- Arthur, L.J., 1988: *Software Evolution: A Software Maintenance Challenge*, Wiley, New York.
- Australian Public Service Commission, 2001: *The Human Resource Capability Model*, <http://www.apsc.gov.au/HRCDP/index.html> [25 April 2007].
- Azuma, M., Mole, D., 1994: Software Management Practices and Metrics in the European Community and Japan: Some Results of a Survey, *Journal of Systems Software*, 26, 1, 5–18.
- Bajers, F., 1998: How to Introduce Maturity in Software Change Management, Technical Report no. 6, June, pp. 112–114, Prentice-Hall, Upper Saddle River, NJ.
- Barry, T., 2002: Outsourcing and the Capability Maturity Model (CMM), [http://www.keane.com/keane\\_ideas/whitepapers/1176\\_1192.htm](http://www.keane.com/keane_ideas/whitepapers/1176_1192.htm) (available on demand).
- Basili, V., Briand, L., Condon, S., Kim, Y., Melo, W., Valett, J., 1996: Understanding and Predicting the Process Software Maintenance Releases. In *Proceedings of the International Conference on Software Engineering*, IEEE, New York.
- Basque, R., 2004: *Un Itinéraire Fléché vers le Capability Maturity Model Integration*, Dunod, Paris.
- Bell Canada, 1995: DATRIX, A Tool for Software Evaluation, Reference Guide, version 2.3, 1995.
- Bennett, K.H., 2002: Software Maintenance: A Tutorial. In *Software Engineering*, edited by Dorfman and Thayer, IEEE Computer Society Press, Los Alamitos, CA, pp. 289–303.
- Bevans, N., 2000: *Introduction to Usability Maturity Assessment*, Serco Usability Services, UK. [http://www.usability.serco.com/trump/documents/Maturity\\_assessment.ppt.pdf](http://www.usability.serco.com/trump/documents/Maturity_assessment.ppt.pdf) [25 April 2007].

- Boehm, B.W., 1981: *Software Engineering Economics*, Prentice-Hall, Upper Saddle River, NJ.
- Boehm, B.W., 1987: Industrial Software Metrics Top 10 List, *IEEE Software*, 4, 5, September, pp. 84–85.
- Bohner, S.A., 1990: Technology Assessment on Software Reengineering, Technical Report, CTC-TR-90-001P, Chantilly, VA. Contel Technology Center.
- Bollinger, T., McGowan, C., 1991: A Critical Look at Software Capability Evaluations, *IEEE Software*, July, 25–41.
- Booch, G., Bryan, D., 1994: *Software Engineering with Ada*, Third Edition, Benjamin/Cummings, Redwood City, CA.
- Bouman, J., Trienekens, J., Van der Zwan, M., 1999: Specification of Service Level Agreements, Clarifying Concepts on the Basis of Practical Research. In *Proceedings of Software Technology and Engineering Practice '99*.
- British Telecommunications, 1990: *Telstar, Software Development Methodology, Pocket Reference Guide, UK*, Release II.1.
- Brito, E., Abreu, F., 1999: Avaliação da Maturidade do Processo, Qualidade no Software, FCT/UNL e INESC.
- Buchman, C., Thomson, H., 1998: The Assessment Model, in SPICE : An Empiricist's Perspective. In *Proceedings of the Second IEEE International Software Engineering Standards Symposium*, pp. 139–169.
- Burnstein, I., Suwannasart, T., Carlson, C., 1996a: Software Test Process Evaluation and Improvement. In *Test Conference Proceedings International*, October 20–25, 581–589.
- Burnstein, I., Suwannasart, T., Carlson C., 1996b: Developing a Testing Maturity Model: Part II, *Crosstalk*, September, 9. <http://www.improveqs.nl/pdf/Crosstalk%20TMM%20part%202.pdf> [25 April 2007].
- Business Link 2007: Computer Software: The Basics, NCC UK Government, <http://www.businesslink.gov.uk/bdotg/action/detail?type=RESOURCES&itemId=1073791279> [November 16 2007].
- Camélia, 1994: *Modèle d'Évolution des Processus de Développement, Maintenance et d'exploitation de Produits Informatiques. Version 0.5*. Projet France-Québec, Montréal, Canada.
- Carey, D., 1994: Executive Round-Table on Business Issues in Outsourcing: Making the Decision. *CIO Canada*, 2(3), 20–29.
- Central Computer and Telecommunications Agency, 2003c: *Application Management*. Information Technology Infrastructure Library (ITIL). Controller of Her Majesty's Stationery Office: Norwich, England.
- Chan, T., Ho, T., 1996: An Economic Model to Estimate Software Rewriting and Replacement Times, *IEEE Transaction on Software Engineering*, 22, 8, 580–598.
- Chapin, N., Hale, J., Khan, K., Ramil, J., Tan, W., 2001: Types of Software Evolution and Software Maintenance, *Journal Software Maintenance Evolution: Research Practice*, 13 1, pp. 3–30.
- Coallier, F., Mayrand, F., Lagüe, B., 1999: Risk Management In Software Product Procurement. In *Elements of Software Process Assessment and Improvement*, IEEE Computer Society Press, Los Alamitos, CA, pp. 23–44.
- Colter, M., 1987: The Business of Software Maintenance. In *Proceedings of 1st Workshop on Software Maintenance*, University of Durham, Durham, England.

- Crawford, J.K., 2002: *Project Management Maturity Model, Providing a Proven Path to Project Management Excellence*. Marcel Dekker/Center for Business Practices, New York.
- Crosby, P.B., 1979: *Quality is Free: The Art of Making Quality Certain*, McGraw-Hill, New York.
- Curtis, B., 1979: In Search of Software Complexitys. In *Proceedings of IEEE/PINY Workshop on Quantitative Software Models*, IEEE Catalog No. TH0067-9, 1979, pp. 95–106.
- Curtis, B., Hefley, W., Miller, S., 1995: People Capability Maturity Model, Software Engineering Institute, CMU/SEI-95-MM-02.
- Dache, G. 2001: IT Companies Will Gain Competitive Advantage by Integrating CMM with ISO9001, *Quality Systems Update*, 11, 11, November.
- DeBaud, J.M., Moopen, B.M., Rugaber, S., 1994: Domain Analysis and Reverse Engineering. In *Proceedings of the International Conference on Software Maintenance*, Victoria, BC, Canada, September 19–23, pp. 326–335.
- Dekleva, S.M., 1992: Delphi Study of Software Maintenance Problems. In *Proceedings of the IEEE Conference on Software Maintenance (ICSM 1992)*. IEEE Computer Society, Orlando, FL, pp. 10–17.
- DeMarco, T., Lister, T., 1985: Programmer Performance and the Effects of the Workplace. In *Proceedings of the 8th International Conference on Software Engineering*.
- Deming, W.E., 1986: *Out of the Crisis*, MIT Press, Cambridge, MA.
- Derider, D., 2002: A Concept-Oriented Approach to Support Software Maintenance and Reuse Activities. In *Proceedings of Workshop on Knowledge-Based Object-Oriented Software Engineering*, 16th European Conference on Object-Oriented Programming (ECOOP 2002), Málaga, Spain.
- Desharnais, J.-M., Paré, F., St-Pierre, D., 1997: Implementing a Measurement Program in Software Maintenance—an Experience Report Based on Basili's Approach. In *IFPUG Conference*, Cincinnati, OH.
- DESMET, 1994: Guidelines for Evaluation Methods Selection, DES/WP2.2/7, Deliverable 2, Version 2.0, NCC Services Ltd., June.
- Dias, M.G., Anquetil, N., Oliveira, K.M., 2003: Organizing the Knowledge Used in Software Maintenance, *Journal of Universal Computer Science*, 9, 7, 641–658.
- Dorfman, M., Thayer, R.H., 1997: *Software Engineering*. IEEE Computer Society Press, Los Alamitos, CA.
- Dorfman, M., Thayer, R.H., 2002: *Software Engineering*, Second Edition, Volume 2—*The Supporting Processes*, edited by Richard H. Thayer and Mark Christensen, IEEE Computer Society Press, Los Alamitos, CA.
- Dove, R., Hartman, S., Benson, S., 1996: *A Change Proficiency Maturity Model—An Agile Enterprise Reference Model with a Case Study of Remmele Engineering*, Agility Forum, New Mexico.
- Duijnhouwer, F.-W., 2003: *Open Source Maturity Model*, Cap Gemini Ernst & Young. [http://www.seriouslyopen.org/nuke/html/modules/Downloads/osmm/GB\\_Expert\\_Letter\\_Open\\_Source\\_Maturity\\_Model\\_1.5.1.pdf](http://www.seriouslyopen.org/nuke/html/modules/Downloads/osmm/GB_Expert_Letter_Open_Source_Maturity_Model_1.5.1.pdf) [25 April 2007].
- Dumke, R., Schmietendorf, A., Zuse, H., 2005: *Formal Descriptions of Software Measurement and Evaluations*, Preprint, Fakultät für Informatik, University of Magdeburg.
- Dutta, S., Van Wassenhove, L., Kulandaiswamy, S., 1998: Benchmarking European Software Management Practices, *Communications of the ACM*, 41, 6.

- Earthy, J., 1998: Usability Maturity Model: Processes, Process Contracting Limited, version 1.2, 27/12/98. Now updated to ver. 2.2. <http://www.idemployee.id.tue.nl/g.w.m.rauterberg/lecturenotes/Usability-Maturity-Model%5B2%5D.PDF> [25 April 2007].
- Ebert, C., Dumke, R., 2004: *Best Practice in Software Measurement*. Springer, Berlin.
- El Emam, K., Drouin, J-N., Melo, W., 1998a: *SPICE: The Theory and Practice of Software Process Improvement and Capability Determination*. IEEE Computer Society Press, Los Alamitos, CA.
- El Emam, K., Goldenson, D.R., 1998b: *Empirical Evaluation of SPICE, in SPICE: The Theory and Practice of Software Process Improvement and Capability Determination*, pp. 287–305. IEEE Computer Society Press, Los Alamitos, CA.
- Emmons, B. 2000: Function Point Maturity Model: How FPs Support the Maturity Model. Presentation to IFPUG, IFPUG 2000 Conference.
- Erlikh, L., 2000: Leveraging Legacy System Dollars for E-business, *IT Pro*, May/June, 17–23.
- European Commission, 1993: *Bootstrap: Global Questionnaire*, Esprit Project #5441, European Commission, Brussels, Belgium.
- European Software Institute (ESI), 1998a: *Terminology Guide*. <http://www.esi.es> (available through ESI partners only).
- European Software Institute (ESI), 1998b: *BaseII, Building a Software Engineering Infrastructure for Improvement and Innovation Using EQFM Model for Business Excellence*.
- European Software Institute (ESI), 1998c: *TeleSpice and R-Spice* (models removed from ESI portfolio during 2002).
- Federal Aviation Administration, 1999: FAM, Federal Aviation Administration Integrated Capability Maturity Model Appraisal Method, v2.0. [www.faa.gov/about/office\\_org/headquarters\\_offices/aio/business\\_value/icmm/](http://www.faa.gov/about/office_org/headquarters_offices/aio/business_value/icmm/) [25 April 2007].
- Federal Aviation Administration, 2001: The Federal Aviation Administration Integrated Capability Maturity Models (FAA-iCMM), v2, 2001. [http://www.faa.gov/about/office\\_org/headquarters\\_offices/aio/documents/media/FAA-iCMMv2.pdf](http://www.faa.gov/about/office_org/headquarters_offices/aio/documents/media/FAA-iCMMv2.pdf) [November 11, 2007].
- Federal Aviation Administration, 2006: Integrated Capability Maturity Model Appraisal Method, v2.0. [Http://www.faa.gov/about/office\\_org/headquarters\\_offices/aio/documents/media/FAMv2-0-FINAL.pdf](http://www.faa.gov/about/office_org/headquarters_offices/aio/documents/media/FAMv2-0-FINAL.pdf) [November 16, 2007].
- Federal Information Processing Standards Publications, 1984: Guideline on Software Maintenance, July, pp. 106–1984.
- Feiler, P. H., Humphrey, W. S., 1992: Software Process Development and Enactment: Concepts and Definitions, CMU/SEI-92-TR-4, Software Engineering Institute, Carnegie Mellon University, Pittsburgh.
- Finkelstein, 1992: A Software Process Immaturity Model. In *Association for Computer Machinery SIGSOFT International Symposium on the Foundations of Software Engineering*, Vol. 17, No. 4, October, 1992, pp. 22–23.
- Fornell, G.E., 1992: Cover Letter to Draft Report, Process for Acquiring Software Architecture, July 10.
- Fowler, M., 1999: *Improving the Design of Existing Code*, Addison-Wesley, Reading, MA.
- Fuggetta, A., Wolf, A., 1996: *Software Process*, Wiley, New York.
- Fusaro, P., El Emam, K., Smith, R., 1997: The Internal Consistencies of the 1987 SEI Matu-

- rity Questionnaire and the SPICE Capability Dimension, International Software Engineering Research Network Technical Report ISERN-97-01 (revised).
- Garcia, S., 1993: Principles of Maturity Modeling. In *Proceedings of SEI Symposium*, Pittsburgh, August, pp. 23–26.
- Garcia Diez, A.B., Lerchundi, R., 1999: TeleSpice Process Model, *ESI White Paper*, European Software Institute (ESI-1999-telespice-ext), December.
- Gilb T. 1995: What We Fail to Do in Our Current Testing Culture, *Testing Techniques Newsletter*, <http://www.soft.com/News/QTN-Online/> (available on demand in the January 1995 issue).
- Glass, R.L., 1992: *Building Quality Software*, Prentice-Hall, Englewood Cliffs, NJ.
- Grady, R., Caswell, D., 1987: *Software Metrics: Establishing a Company-wide Program*, Prentice-Hall, Englewood Cliffs, NJ.
- Grant, G., 1997: IS Administration Course, Implication of the IS Capability Model, Course 273-432/636, McGill University, Montréal, December.
- Gaydon, A.W., Nevalainen, R., 1998: The Reference Model, in SPICE: An Empiricist's Perspective. In *Proceedings of the Second IEEE International Software Engineering Standards Symposium*, 75–97.
- Grubb, P., Takang, A., 2003: *Software Maintenance Concepts and Practice*, Second Edition, World Scientific, Singapore.
- Halstead, M.H., 1978: Software Science: A Progress Report. In *Proceedings, U.S. Army/IEEE Second Software Life Cycle Management Conference*, Atlanta, GA, August 1978, pp. 174–179.
- Harel, D., Politi, M., 1998: *Modeling Reactive Systems with State Charts: The Statemate Approach*, McGraw-Hill, New York.
- Harrison, W., Cook, C., 1990: Insights on Improving the Maintenance Process Through Software Measurement. In *Proceedings of the International Conference on Software Maintenance*, San Diego, November, pp. 37–44.
- Huffman-Hayes, J., Patel, S., Zhao, L., 2004: A Metrics-Based Software Maintenance Effort Model. In *Proceedings of the 8th European Conference on Software Maintenance and Reengineering, CSMR 2004*, IEEE Computer Society, Tampere, Finland, pp. 254–258.
- Hayes, S., 1994: Project Management: How It Can Improve The Maintenance Process, *Application Development Trends*, October 1994.
- Hopkinson, J.P., 1996: System Security Engineering Maturity Model, EWA-Canada. <http://www.sse-cmm.org/docs/sse-cmm.pdf> [25 April 2007].
- Humphrey, W., 1990: *Managing the Software Process*, Software Engineering Institute, Addison-Wesley, Reading, MA.
- Humphrey,W., Kitson, D., Gale, J., 1991: A Comparison of U.S. and Japanese Software Process Maturity. In *Proceedings of the 13th International Conference on Software Engineering*, Austin, May, pp. 38–49.
- Ibrahim, L., 1998: The Federal Aviation Administration Integrated Capability Maturity Models (FAA-CMMi), Smart Buying with the FAA's ICMM, *Crosstalk*, 11, 9, pp. 15–19. <http://www.stsc.hill.af.mil/crosstalk/1998/11/ibrahim.asp> [25 April 2007].
- Ibrahim, R., Hirmanpour, I., 1995: The Subject Matter of Process Improvement: A Topic and Reference Source for Software Engineering Educators and Trainers, CSMU/SEI-95-TR-003, Software Engineering Institute, Carnegie Mellon University, Pittsburgh.

- IEC, 1994: *Reliability and Maintainability Management*, Second Edition, IEC Standard Publication 300, Geneva.
- Institute of Electrical and Electronics Engineers, 1990: IEEE Standard Glossary of Software Engineering Terminology, Standard 610.12-1990, Institute of Electrical and Electronics Engineers, New York.
- Institute of Electrical and Electronics Engineers, 1997: IEEE Standard for Developing Software Life Cycle (Software), Standard 1074-1997, Institute of Electrical and Electronics Engineers, New York.
- Institute of Electrical and Electronics Engineers, 1998a: IEEE Standard for Software Maintenance, Standard 1219, Institute of Electrical and Electronics Engineers, New York.
- Institute of Electrical and Electronics Engineers, 1998b, IEEE Standard for a Software Quality Metrics Methodology, Standard 1061-1998, Institute of Electrical and Electronics Engineers, New York.
- Institute of Electrical and Electronics Engineers, 1998c: IEEE Guide for the Use of the IEEE Standard Dictionary of Measures to Produce Reliable Software, Standard 982.2-1988, Institute of Electrical and Electronic Engineers, New York.
- International Organisation for Standardization, 1995: ISO/IEC Standard 12207. *Standard for Information Technology: Software Lifecycle Processes*. International Organisation for Standardization/International Electrotechnical Commission, Geneva.
- International Organisation for Standardization, 2001: *Software Engineering—Product Quality. Part 1: Quality Model*, ISO/IEC Standard 9126. International Organisation for Standardization—International Eletrotechnical Commission, Geneva.
- International Organisation for Standardization, 2002: ISO/IEC 15939. *Practical Software and System Measurement*, ISO/IEC 15939:2002, International Organisation for Standardization, Geneva.
- International Organisation for Standardization, 2002a: *Guidelines for the Quality and/or Environmental Management System Auditing*, ISO19011:2002, International Organisation for Standardization, Geneva.
- International Organisation for Standardization, 2003: *Information Technology—Process Assessment. Part 2: Performing an Assessment*, ISO/IEC 15504-2:2003. International Organisation for Standardization/International Electrotechnical Commission, Geneva.
- International Organisation for Standardization, 2004a: *Software Engineering: Guidelines for the Application of ISO9001:2000 to Computer Software*, ISO/IEC Standard 90003:2004. International Organisation for Standardization/International—Electrotechnical Commission, Geneva.
- International Organisation for Standardization, 2004b: *Information Technology: Process Assessment—Part 1: Concepts and Vocabulary*, ISO/IEC 15504-1:2004. International Organisation for Standardization/International—Electrotechnical Commission, Geneva.
- International Organisation for Standardization, 2004c: *Information Technology: Process Assessment—Part 3: Guidance on Performing Assessments*, ISO/IEC 15504-3:2004. International Organisation for Standardization/International—Electrotechnical Commission, Geneva.
- International Organisation for Standardization, 2004d: *Information Technology: Process Assessment—Part 4: Guidance on Use for Process Improvement and Process Capability Determination*, ISO/IEC 15504-4:2004. International Organisation for Standardization/International—Electrotechnical Commission, Geneva.

- International Organisation for Standardization, 2004e: ISO/IEC 17011. *Conformity Assessment, General Requirements for Accreditation Bodies Accrediting Conformity Assessment Bodies*, ISO/IEC 17011:2004, International Organisation for Standardization, Geneva.
- International Organisation for Standardization, 2006: *Standard for Information Technology: Software Maintenance, ISO/IEC Standard 14764*, International Organisation for Standardization—International Eletrotechnical Comission, Geneva.
- International Software Benchmarking Standards Group, 2007: Data Collection Questionnaire: Application Software Maintenance and Support. Version 1.2.2, International Software Benchmarking Standards Group: Victoria, Australia. <http://www.isbsg.org/isbsg.nsf/weben/Downloads> [November 27, 2007].
- IT Governance Institute, 2007: *COBIT, Governance, Control and Audit for Information and Related Technology*. ISACA: Rolling Meadows, Illinois, Release 4.1. [http://www.isaca.org/Content/NavigationMenu/Members\\_and\\_Leaders/COBIT6/Obtain\\_COBIT/Obtain\\_COBIT.htm](http://www.isaca.org/Content/NavigationMenu/Members_and_Leaders/COBIT6/Obtain_COBIT/Obtain_COBIT.htm) [25 April 2007].
- ITIL (Office of Government Commerce. Information Technology Infrastructure Library), 2007a: Service Strategy, Controller of Her Majesty's Stationery Office, Norwich, UK.
- ITIL (Office of Government Commerce. Information Technology Infrastructure Library), 2007b: Service Design, Controller of Her Majesty's Stationery Office, Norwich.
- ITIL (Office of Government Commerce. Information Technology Infrastructure Library), 2007c: Service Transition, Controller of Her Majesty's Stationery Office, Norwich, UK.
- ITIL (Office of Government Commerce. Information Technology Infrastructure Library), 2007d: Service Operation, Controller of Her Majesty's Stationery Office, Norwich, UK.
- ITIL (Office of Government Commerce. Information Technology Infrastructure Library), 2007e: Continual Service Improvement, Controller of Her Majesty's Stationery Office, Norwich, UK.
- Jarzabek, S., 2007: *Effective Software Maintenance and Evolution*, Auerback Publications, Boca Raton, FL.
- Jones, C., 1994: *Assessment and Control of Software Risks*, Prentice-Hall, Upper Saddle River, NJ.
- Kajko-Mattsson, M., 2001a: *Corrective Maintenance Maturity Model*, partial fulfilment of the requirements for Ph.D., report 01-015, Stockholm University.
- Kajko-Mattsson, M., Forssander, S., Westblom, U., 2001b: Corrective-Maintenance Maturity Model (Cm3): Maintainer's Education and Training ICSE, pp. 610–619.
- Kajko-Mattsson M., Ahnlund, C., Lundberg, E., 2004: CM3: Service Level Agreement. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2004)*. Chicago, pp. 432–436.
- Kajko-Mattsson, M., Forssander, S., Olsson, U., 2001: Corrective Maintenance Maturity Model (Cm3): Maintainer's Education and Training. In *Proceedings of the 23rd International Conference on Software Engineering*, May, Toronto, pp. 610–619.
- Karten, N., 2007: Establishing Service Level Agreements. Karten Associates. <http://www.nkarten.com/slaservices.html> [25 April 2007].
- Kerzner H., 2002: *Strategic Planning for Project Management Using a Project Management Maturity Model*. Wiley, New York.
- Khan, K., Zhang, Y., 2005: *Managing Corporate Information Systems Evolution and Maintenance*. Idea Group, IGI Global, Hershey, PA.

- Kitchenham, B., Guilherme, H., et al., 1999: Towards an Ontology of Software Maintenance, *Journal Software Maintenance: Research and Practice*, 11, pp. 365–389.
- Koltun, P., Hudson, A., 1991: A Reuse Maturity Model, In *Proceedings of the Fourth Annual Workshop on Software Reuse*, L. Latour, Editor, Department of Computer Science, University of Maine, Orono, ME 04469, pp. 1–4.
- Koskinen, J., Ahonen, J.J., Sivula, H., Tulus, T., Lintinen, H., Kankaanpää, I., 2005a: Software Modernization Decision Criteria—An Empirical Study. In *The Ninth European Conference on Software Maintenance and Reengineering (CSMR 2005)*. Manchester, UK, March.
- Koskinen, J., Salminen, A., 2005b: Supporting Impact Analysis in HyperSoft and Other Maintenance Tools. In *Twenty-Third IASTED International Multi-Conference on Applied Informatics (SE 2005)*, Innsbruck, Austria, February, pp. 187–192.
- Krause, M.H., 1994: Software—A Maturity Model for Automated Software Testing. *Medical Devices and Diagnostic Industry Magazine*, December, p. 8. <http://www.devicelink.com/mddi/archive/94/12/014.html> [25 April 2007].
- Kubicki, C., 1993: The System Administration Maturity Model. In *Proceedings of the Seventh System Administration Conference (LISA'93)*, Monterey, California, November. <http://www.usenix.org/publications/library/proceedings/lisa93/kubicki.html> [November 16, 2007].
- Kwack, Y. H., Ibbs, C. W., 1997: Project Management Maturity (PM)<sup>2</sup> Model. <http://www.ce.berkeley.edu/~ibbs/yhkwak/pmmaturity.html> [December 1, 2007].
- Laguë, B., April, A., 1996: Mapping of the ISO9126 Maintainability Internal Metrics to an Industrial Research Tool. In *Proceedings of SES 1996*, Montreal, October 21–25. <http://www.lrgl.uqam.ca/sponsored/ses96/paper/lague.html> [25 April 2007].
- Layzell, P., Macaulay, L. 1990: An Investigation into Software Maintenance Perception and Practices. In *Proceedings IEEE International Conference on Software Maintenance*, pp. 130–140.
- Lehman, M.M., 1980: Programs, Life-Cycles and the Laws of Software Evolutions, *Proceedings of the IEEE*, 68, 9, 1060–1076.
- Lientz, B., Swanson, E., 1980: *Software Maintenance Management*, Addison-Wesley, Reading, MA.
- Liso, A., 2001: Software Maintainability Metrics Model: An Improvement on the Coleman-Oman Model, *Crosstalk*, August, 15–17.
- Luftman J., 2001: Assessing Business-IT Alignment Maturity. *Communications of the Association for Information Systems*, 4, 14, 51. <http://www.dme.uma.pt/caise06dc/papers/Tapia.pdf> [password required, 25 April 2007].
- Maclennan, F., Ostrolenk, G., et al., 1998: Introduction to the SPICE Trials, in SPICE: An Empiricist's Perspective. In *Proceedings of the Second IEEE International Software Engineering Standards Symposium*, pp. 269–286.
- Madhavji, N., Hoeltje, D., Hong, W., Bruchhaus, T., 1994: Elicit: A Method for Eliciting Process Models. In *Proceedings of the Third International Conference on the Software Process*, pp. 111–122, IEEE Computer Society Press, Los Alamitos, CA.
- Malcolm Baldrige National Quality Program, 2007: *Criteria for Performance Excellence*, NIST. <http://www.quality.nist.gov/>.
- Martin, J., McClure, C., 1983: *Software Maintenance: The Problem and Its Solutions*. Prentice-Hall, Englewood Cliffs, NJ.

- Maya, M., Abran, A., Bourque, P., 1996: Measuring the Size of Small Functional Enhancement to Software. In *6th International Workshop on Software Metrics*, University of Regensburg, Germany.
- Mayrand, J., 1995: *Software Engineering Bibliography on Software Assessment Using Static Source Code Analysis*, École Polytechnique de Montréal, Montréal, Québec, September.
- McBride, D. 1995: Successful Deployment of IT Service Management in the Distributed Enterprise. In *21st International Computer Measurement Group Conference*, December, Nashville, pp. 1036–1045.
- McCabe, T.J., 1976: A Complexity Measure, *IEEE Transactions on Software Engineering, SE-2*, 4, November, 308–320.
- McClure, C.L., 1997: *Managing Software Development and Maintenance*. Van Nostrand Reinhold, New York.
- McCracken B., 2002: *Taking Control of IT Performance*, InfoServer LLC, Dallas, October. <http://www.outsourcing-information-technology.com/control.html> [25 April 2007].
- McGarry, J., 1995: Practical Software Measurement: A Guide to Objective Program Insight, Department of Defense, September 1995.
- Menk, C.G., 1996: System Security Engineering Capability Maturity Model (SSE-CMM), Department of Defense. <http://www.sse-cmm.org/lib/lib.asp> [25 April 2007].
- Moore, J.W., 1998: *Software Engineering Standards: A User's Road Map*. IEEE Computer Society Press, Los Alamitos, CA.
- Moore, J.W., 2005: *The Roadmap to Software Engineering: A Standards-Based Guide*, Wiley–IEEE Computer Society Press, Hoboken, NJ.
- Moriguchi, S., 1996: *Software Excellence: A Total Quality Management Guide*, Productivity Press.
- Mullins C., 2002: The Capability Model—From a Data Perspective. *Data Administration Newsletter*. <http://www.tdan.com/i003fe04.htm> [25 April 2007].
- NASCIO, 2003: NASCIO Enterprise Maturity Model, version 1.3, December 2003. [www.nascio.org/publications/documents/NASCIO-EAMM.pdf](http://www.nascio.org/publications/documents/NASCIO-EAMM.pdf) [25 April 2007].
- Nelson, R.R., Winter, S.G., 1982: *An Evolutionary Theory of Economic Change*, Belknap Press, Cambridge, MA.
- Niessink, F., van Vliet, H., 1999: The Vrije Universiteit IT Service Capability Maturity Model, technical report IR-463, release L2-1.0. <http://www.cs.vu.nl/~hans/publications/y1999/ITSCMM-VUrapport.pdf> [30 November 2007].
- Niessink, F., 2000: Perspectives on Improving Software Maintenance, SIKS Ph.D. dissertation, Dutch Graduate School for Information and Knowledge Systems.
- Niessink, F., Clerk, V., van Vliet, H., 2005: *The IT Service Capability Maturity Model*, 1.0 release candidate 1, Software Engineering Research Centre, Utrecht, The Netherlands. <http://www.itservicecmm.org/> [25 April 2007].
- Nord, W.R., Tucker, S., 1987: *Implementing Routine and Radical Innovations*, Lexington Books, Lexington, MA.
- Osborne, W.M., Chikofsky, E.J., 1990: Fitting Pieces to the Maintenance Puzzle, *IEEE Software*, 7, 1, 10–11.
- Paquette, D-A., April, A., Abran, A., 2006: Assessment Results Using the Software Maintenance Maturity Model (S<sup>3</sup>m). In *Proceedings of the 16th International Workshop on Software Measurement* (IWSM-Metrikom 2006), Postdam, Germany, pp. 147–160 .

- Parikh, G., Kajko-Mattsson, M., 2002: Are We Making Enough Progress within Software Maintenance? *Software Dioxide*, <http://www.softwaredioxide.com/Channels/ConView.asp?id=7155> [25 April 2007].
- Paulk, M., 1995: The Evolution of the SEI's Capability Maturity Model for Software, *Software Process*, 1, August, 50–60.
- Paulk, M.C., Curtis, B., 1991: Capability Maturity Model for Software, CMU/SEI-91-TR-24, ADA240603, Software Engineering Institute, Carnegie Mellon University, Pittsburgh.
- Paydarfar, S., 2001: An Integration Maturity Model for the Digital Enterprise, MCS Software Corporation [http://www.mscsoftware.com/support/library/journal/sum2001\\_maturitymodel.pdf](http://www.mscsoftware.com/support/library/journal/sum2001_maturitymodel.pdf) [30 November 2007].
- Pfleeger, S.L., 2001: *Software Engineering—Theory and Practice*, Second Edition, Prentice-Hall: Englewood Cliffs, NJ.
- Pigoski, T.M., 1994: *Software Maintenance*, *Encyclopedia of Software Engineering*, Wiley, New York.
- Pigoski, T.M., 1997: *Practical Software Maintenance: Best Practice for Managing Your Software Investment*, Wiley, New York.
- Pigoski, T.M., April, A., 2005: *Software Maintenance*, Chapter 6, “Ironman Version of the Guide to the Software Engineering Body of Knowledge,” pp. 6-1–6-15, IEEE Computer Society Press, Los Alamitos, CA.
- Polo, M., Piattini, M., Ruiz, F., 2002: *Maintenance Management: Technologies and Solutions*, Idea Group Publishing, ICI Global, Hershey, PA.
- Pressman, R.S., 1992: *Software Engineering: A Practitioner's Approach*. Third Edition. McGraw-Hill, New York.
- Pressman, R.S., 2001: *Software Engineering: A Practitioner's Approach*, Fourth Edition, McGraw-Hill, New York.
- Pressman, R.S., 2004: *Software Engineering: A Practitioner's Approach*. Sixth Edition. McGraw-Hill, New York.
- Raffoul, W., 2002: *The Outsourcing Maturity Model*, Meta Group, <http://techupdate.zdnet.com/techupdate/stories/main/0%2C14179%2C2851971-2%2C00.html#level1> [25 April 2007].
- Rayner, P., Reiss, G., 2001: The Programme Management Maturity Model. The Programme Management Group. [http://www.12manage.com/methods\\_reiss\\_pmmm.html](http://www.12manage.com/methods_reiss_pmmm.html) [November 16, 2007].
- Ream, S.W., 2003: *The Business Continuity Maturity Model*, Virtual Corporation, [http://www.virtual-corp.net/html/business\\_continuity.html](http://www.virtual-corp.net/html/business_continuity.html) [25 April 2007].
- Rogers, E.M., 1995: *Diffusion of Innovations*, Free Press, New York.
- Ruiz F., Vizcaíno Barceló, A., Piattini, M., García, F., 2004: An Ontology For The Management Of Software Maintenance Projects. *International Journal of Software Engineering and Knowledge Engineering* 14, 3, 323–349.
- Sahin I., Zahedi, M., 2001: Policy Analysis for Warranty, Maintenance and Upgrade of Software Systems, *Journal of Software Maintenance: Research and Practice*, 13, 469–493.
- Satriani, G., 1997: A European Software Best Practice Repository, *Software Process: Improvement and Practice*, 3, 4, 243–245
- Schekkerman, J., 2003: *Extended Enterprise Architecture Maturity Model*, Institute for Enterprise Architecture Development, <http://www.enterprise-architecture.info/Images/Ex>

- tended%20Enterprise/Extended%20Enterprise%20Architecture.htm#e2amm [25 April 2007].
- Scheuing, A.Q., Fruhauf, K., 2001: Maturity Model for IT Operations (MITO), <http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/9013/28613/01281425.pdf?arnumber=1281425> [25 April 2007].
- Schlichter, J., 2002: An Introduction to the Emerging PMI Organisational Project Management Maturity Model, [http://www.pmi.org/prod/groups/public/documents/info/pp\\_opm3.asp](http://www.pmi.org/prod/groups/public/documents/info/pp_opm3.asp) [25 April 2007].
- Schmidt, M., 2000: *Implementing the IEEE Software Engineering Standards*, Sams Publishing, Indianapolis, Indiana.
- Schmietendorf, A., Scholz, A., 1999: The Performance Engineering Maturity Model at a Glance, *Metrics News*, 4, 2, 342.
- Schneidewind, N.F., 1987: The State of the Maintenance, *IEEE Transactions on Software Engineering*, 13, 3, 303–310.
- Schorsch, T., 1996: The Capability Imm-Maturity Model, *Crosstalk*, November. <http://www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1996/11/xt96d1h.asp> [25 April 2007].
- Seacord, R., Plakosh, D., Lewis, G., 2003: *Modernizing Legacy Systems: Software Technologies, Engineering Process, and Business Practices*, Addison-Wesley, Reading, MA.
- Sellami, A., 2001: Analyse Comparative des Modèles de Maintenance du Logiciel entre SWEBOK, ISO/IEC 14764 et la Littérature, *Mémoire de Maîtrise*, UQAM, April.
- Sharpley, W.K., 1977: Software Maintenance Planning for Embedded Computer Systems. In *Proceedings of the IEEE COMPSAC*, November, 520–526.
- Sheard, S., 1997: The Frameworks Quagmire, Software Productivity Consortium. <http://www.stsc.hill.af.mil/crosstalk/1997/09/frameworks.asp> [November 16, 2007].
- Sink, D.S., 1985: *Productivity Management: Planning, Measurement and Evaluation, Control and Improvement*, Wiley, New York.
- Sivaguru, S., Murthy, K., 1994: *A TQM Roadmap for ISO9001/SEI CMM Organisations*, Integra Techsoft Pvt. Ltd., Bangalore India. <http://www.softwaredioxide.com/Channels/conView.asp?id=3773> [25 April 2007].
- Software Engineering Institute, 1993a: Evolutionary Model Practices for CCM, version 1.1, CMU/SEI-93-TR-25, ESC-TR-93-177. Software Engineering Institute, Carnegie Mellon University, Pittsburgh.
- Software Engineering Institute, 1993b: Capability Maturity Model for Software (CMM), version 1.1, CMU/SEI-93-TR-24, ESC-TR-93-177. Software Engineering Institute, Carnegie Mellon University, Pittsburgh.
- Software Engineering Institute, 1995: System Engineering Capability Maturity Model (SE-CMM), version 1.1, SECMM-95-01, CMU/SEI-95-MM-003. Software Engineering Institute, Carnegie Mellon University, Pittsburgh.
- Software Engineering Institute, 1996a: SE-CMM Appraisal Method, version 1.1, CSMU/SEI-96-HB-004, Software Engineering Institute, Carnegie Mellon University, Pittsburgh.
- Software Engineering Institute, 1996b: Software Capability Evaluation (SCE): Method Description, version 3.0, CMU/SEI-96-TR-002. Software Engineering Institute, Carnegie Mellon University, Pittsburgh.
- Software Engineering Institute, 2000: Standard CMMi Appraisal Method for Process Im-

- provement (SCAMPI): Method Description, version 1.0, CMU/SEI-2000-TR-009. Software Engineering Institute, Carnegie Mellon University, Pittsburgh.
- Software Engineering Institute, 2001a: Appraisal Requirements for CMMi (ARC), Version 1.1, CMU/SEI-2001-TR-034. Software Engineering Institute, Carnegie Mellon University, Pittsburgh.
- Software Engineering Institute, 2001b: CMM Based Appraisals for Internal Process Improvement (CBA IPI), version 1.2, CMU/SEI-2001-TR-033. Software Engineering Institute, Carnegie Mellon University, Pittsburgh.
- Software Engineering Institute, 2002: Capability Maturity Model Integration for Software Engineering (CMMi), version 1.1, CMU/SEI-2002-TR-028. Software Engineering Institute, Carnegie Mellon University, Pittsburgh.
- Software Productivity Consortium, 1992: *Process Definition and Modeling Handbook*, SPC, Herndon, VA.
- Sommerville, I.; Sawyer, P., 1997: *Requirements Engineering: A Good Practice Guide*, Wiley, Chichester.
- Sribar, V.; Vogel, D., 2001: The Capability Maturity Model for Operations. Metagroup, acquired by Gartner on 1 April 2005, Continue to Metagroup.com: <http://www.metagroup.com/metaview/mv0452/mv0452.html> [April 11, 2005].
- Staab, T.C., 2002: Using the SW-TMM to Improve Testing Process, *Crosstalk*, November, pp. 13–16.
- Stark, G.E., Kern, L.C., Vowell, C.V., 1994: A Software Metric Set for Programme Maintenance Management, *Journal of Systems and Software*, March, 239–249.
- Stark, G.E.; Oman, P., 1997: Software Maintenance Management Strategies: Observations from the Field. *Journal of Software Maintenance: Research and Practice*, 99, 6, 365–378
- St-Pierre, D., 1993: Integration of Maintenance Metrics. In *IEEE International Conference on Software Maintenance—ICSM 1993*. Montréal, Québec, pp. 374–375.
- Stratton, R.W.: The earned value management maturity model. <http://www.mgmt-technologies.com/evmtech.html>, 2000 [25 April 2007].
- Swanson, E.B., 1976: The Dimensions of Maintenance. In *Proceedings of the 2nd International Conference on Software Engineering*, pp. 492–497. October, San Francisco.
- Swanson, E.B., Beath, C.M., 1989: *Maintaining Information Systems in Organizations*. Wiley, New York.
- Tajima, D., Matsubara, T., 1981: The Computer Software Industry in Japan, *Computer*, May.
- Thompson, P., 1992: The Twelve Best Practices in Software Maintenance, white paper, 2004, [www.risglobal.com/Publications/BestPracticesInASM.pdf](http://www.risglobal.com/Publications/BestPracticesInASM.pdf) [25 April 2007].
- TICKIT Project Office, 2001: *Guide to Software Quality Management System Construction and Certification Using EN29001, Issue 5*, UK Department of Trade and Industry and BCS, UK.
- Tobia, E., Glynn, M., 2001: E-business, Can We Control It? E-business Maturity Model. In *14th Utility Coal Conference*, Price Waterhouse Coopers, New York.
- Topaloglu, Y., Dikenelli, O., Sengonca, H., 1998: A Four Dimensional Reuse Maturity Model. In *Symposium on Computer and Information Sciences*, [www.ie.inf.uc3m.es/grupo/Investigacion/LineasInvestigacion/Congresos/RMM97\\_Docum\\_Final.doc](http://www.ie.inf.uc3m.es/grupo/Investigacion/LineasInvestigacion/Congresos/RMM97_Docum_Final.doc) [25 April 2007].
- Tornatzky, L.G.; Fleischner, M., 1990: *The Process of Technological Innovation*, Lexington Books, Lexington, MA.

- Trillium, 1991: Model for the Telecom Product Development & Support Process Capability, Bell Canada, version 1.0, 91/08.
- Trillium, 1994: Model for the Telecom Product Development & Support Process Capability, Bell Canada, version 3.0, 1994, <http://www2.umassd.edu/swpi/BellCanada/trillium-html/trillium.html> [November 11, 2007].
- U.S. Department of Commerce, 2003: IT Architecture Capability Maturity Model (ACMM). Currently version 8.1.1 <Http://www.opengroup.org/architecture/togaf8-doc/arch/p4/maturity/mat.htm> [November 30, 2007].
- Van Bon, J. (Ed.), 2000: *World Class IT Service Management Guide 2000*, ten Hagen & Stam Publishers, Amsterdam, The Netherlands.
- van Herwaarden, C.J., Grift, F.U., 1999: IPW & IPW Stadia Model, version 1.0, Quint Wellington Redwood, Amsterdam, The Netherlands.
- van Zuylen, H., 1993: *The REDO Compendium: Reverse Engineering for Software Maintenance*, Wiley, New York.
- Veenendaal, V., Swinkels, R., 2002: Guideline for Testing Maturity: Part 1: The TMM Model, *Professional Tester*, 3, 1. <http://www.tmmi.nl/pdf/TMM%20testing%20professional%20v1.pdf> [25 April 2007].
- Vetter, R., 1999: The Network Maturity Model for Internet Development, *IEEE Computer*, 132, 10, 117–118.
- Visaggio, G., 2000: Value-Based Decision Model for Renewal Process in Software Maintenance, *Annals of Software Engineering* 9, 215–233.
- Visconti, M., Cook, C., 1992: Software System Documentation Process, Computer Science Department, Oregon State University, Technical Report 92-60-11, <http://web.engr.oregonstate.edu/~cook/doc/Papers.htm> [November 30, 2007].
- Vizcaíno, A., Favela, J., Piattini, M., 2003: *A Multi-Agent System for Knowledge Management in Software Maintenance*, Vol. 2773, Springer, Berlin, pp. 415–421.
- von Mayrhofer, A., 1990: *Software Engineering: Methods and Management*, Academic Press, San Diego CA.
- Wang, Y., King, G., 2001: *Software Engineering Processes—Principles and Applications*. CRC Press, Boca Raton, FL.
- Welker, K., Oman, P.W., Atkinson, G., 1997: Development and Application of an Automated Source Code Maintainability Index, *Journal of Software Maintenance*, May/June, 127–159.
- Welker, K., 2001: The Software Maintainability Index Revisited, *Crosstalk*, August. <http://www.stsc.hill.af.mil/crosstalk/2001/08/welker.html> [November 18, 2007].
- Wichita State University, 1999: Enterprise Engineering Presentation—Capability Model of Business Process Reengineering. Course IE801. Wichita, KS.
- Widdows, C., Duijnhouwer, F-W., 2003: *Open Source Maturity Model*. Cap Gemini Ernst & Young. [http://www.seriouslyopen.org/nuke/html/modules/Downloads/osmm/GB\\_Expert\\_Letter\\_Open\\_Source\\_Maturity\\_Model\\_1.5.1.pdf](http://www.seriouslyopen.org/nuke/html/modules/Downloads/osmm/GB_Expert_Letter_Open_Source_Maturity_Model_1.5.1.pdf) [25 April 2007].
- Windley, P.J.: eGovernment Maturity, CIO State of Utah. 2002. <http://www.windley.com/docs/eGovernment%20Maturity.pdf> [25 April 2007].
- Yau, S.S., Collofello, J.S., MacGregor, T.M., 1978: Ripple Effect Analysis for Software Maintenance. In *Proceedings Second International Computer Software and Applications Conference (COMPSAC '78)*, November, pp. 60–65.
- Yu, E., Mylopoulos, J., 1994: Understanding “Why” in Process Modeling, Analysis and De-

- sign. In *16th International Conference on Software Engineering*, Sorrento, Italy, May, pp. 159–168.
- Zitouni, M.; Abran, A.; Bourque, P., 1995: Élaboration d'un outil d'évaluation et d'amélioration du processus de la maintenance des logiciels: une piste de recherche, Le génie logiciel et ses applications, *Huitièmes Journées Internationales (GL95)* , EC2 & Cie , Paris, La Défense, 727–739.
- Zitouni, M., Abran, A., 1996: A Model to Evaluate and Improve the Quality of the Software Maintenance Process. In *Proceedings of 6th International Conference on Software Quality Conference, ASQC—Software Division*, Ottawa, pp. 238–258.

# Index

- Acceptance criteria, 237
- Acceptance testing, 237
- Acceptance tests road map, 185
- Acceptance/refusal process, 13
- Acquirer, 237
- Acquirer–supplier agreement, 237
- Acquisition, 237
- Acquisition organization, 237
- Acquisition organization’s standard
  - software acquisition process, 237
- Action item, 237
- Action plan road map, 135
- Action proposal, 237
- Activity, 237
- Ad hoc requests/reports/services road map, 178
- Adaptive maintenance, 237
- After-hours support road map, 177
- Age (of a problem), 237
- Agreement, 238
- Allocate, 238
- Allocated baseline, 238
- Analysis of innovation/improvement
  - proposal road map, 147
- Application domain, 238
- Appraisal method, 238
- Appraisal team, 238
- Appraise, 238
- Approved modification architecture, 238
- Artifact, 238
- Assessed capability, 238
- Assessment constraints, 238
- Assessment indicator, 238
- Assessment input, 238
- Assessment instrument, 238
- Assessment output, 238
- Assessment participant, 239
- Assessment process, 203
- Assessment purpose, 239
- Assessment record, 239
- Assessment scope, 239
- Assessment sponsor, 239
- Assessment tool, 203
- Associated documentation, 239
- Associated processes, 239
- Attribute, 239
- Attribute indicator, 239
- Attributes (of software), 239
- Audit, 239
  - client, 239
  - conclusions, 239
  - criteria, 239
  - evidence, 239
  - findings, 239
  - program, 239
  - scope, 239
  - team, 239
- Auditee, 239
- Auditor, 239
- Availability, 240
- Base practice, 240
- Baseline, 240
- Baseline configuration management, 240
- Baseline configuration road map, 189
- Baseline management, 240
- Benchmark, 240
- Benchmarking, 240

- Boundary, 240
- Business rules and functional support road map, 177
- Camélia model, 44
- Capability, 240
- Capability dimension, 240
- Capability evaluation, 240
- Capability level, 240
- Capability maturity model, 240
- Case studies of the use of S3<sup>m®</sup>, 203
- Causal analysis, 241
- Causal Analysis and Problem Resolution KPA, 124
  - expected results of, 125
  - goals of, 125
  - links with other KPAs, 125
- Causal analysis and problem resolution, 195
  - detailed exemplary practices, 195
    - Level 0, 195
    - Level 1, 195
    - Level 2, 196
- Causal analysis meeting, 241
- Cause, 241
- Change agent, 241
- Change authority, 241
- Change control, 241
- Change management, 241
- Change management road map, 189
- Characteristic, 241
- Choosing between ISO 9001 and CMMi, 46
- Classification of software maintenance processes, 73
- CM<sup>3®</sup> Model, 56
- CMM, 45
- CMMi, 44, 45, 46, 48, 51
  - evaluation classes characteristics, 52
  - for services design process, 55
  - for services model, 57
  - model architecture, 58
- Commitment, 241
- Common cause (of a defect), 241
- Common cause of variation, 241
- Common features, 241
- Communicate nonconformance road map, 193
- Communications and contact structure road map, 150
- Compatible assessment model, 242
- Competent assessor, 242
- Completeness, 242
- Compliance, 242
- Compliance article, 242
- Component, 242
- Composite results, 242
- Concession, 242
- Concession; waiver, 242
- Conditional, 242
- Configuration, 242
- Configuration and change management, 188
  - detailed exemplary practices, 188
    - Level 0, 188
    - Level 1, 188
    - Level 2, 189
- Configuration and Version Management KPA, 120
  - expected results of, 122
  - goals of, 121
  - links with other KPAs, 121
- Configuration baseline, 242
- Configuration control, 243
- Configuration identification, 243
- Configuration item, 243
- Configuration management, 243
- Configuration management library system, 243
- Configuration unit, 243
- Conformity, 243
- Consequence, 243
- Consistency, 243
- Constraint, 243
- Constructed capability, 243
- Contingency factor, 243
- Continuous architecture, 244
- Contract, 244
- Contract integrity, 244
- Contract terms and conditions, 244
- Contractor, 244
- Convertibility, 244
- Correction, 244
- Corrective action, 244
- Corrective maintenance, 244
- Cosmetic defect, 243
- Cost, 244
- Critical computer resource, 244
- Critical technical parameter, 244
- Customer, 244

- Customer account management road map, 163  
Customer dissatisfaction, 244  
Customer expectations, 244  
Customer requirements, 245  
Customer satisfaction, 245
- Data, 245  
Data management, 245  
Defect, 245  
Defect density, 245  
Defect prevention, 245  
Defect review, 245  
Defect root cause, 245  
Defined level, 245  
Defined process, 245  
Definition of maintenance measures road map, 145  
Degree of confidence, 246  
Deliverable, 246  
Delivered products, 246  
Dependability, 246  
Dependency item, 246  
Deployment of innovation/improvement road map, 148  
Derived requirement, 246  
Design, 246  
Design and development, 246  
Design authority, 246  
Design recovery, 246  
Design review, 246  
Detailed design baseline, 246  
Detailed design road map, 179  
Developer, 247  
Development baseline, 247  
Development life cycle, 247  
Development phase, 247  
Deviation, 247  
Deviation permit, 247  
Differences between maturity model, quality standards, and evaluation method, 46  
Differences between operations, development, and maintenance, 11  
Direct measure, 247  
Direct measurement method, 247  
Discrimination, 247  
Disposition, 247  
Document, 247
- Documentation, 247  
Documentation and standardization of processes/services road map, 137  
Documentation profile, 247  
Documentation road map, 181  
Domain, 247
- Effective, 247  
Effective process, 247  
Effectiveness, 248  
Effectiveness assessment, 248  
Efficiency, 248  
Effort, 248  
Empowerment, 248  
Enabling system, 248  
End system, 248  
End user, 248  
End user representatives, 248  
Engineering group, 248  
Engineering plan, 248  
Enhanced capability, 248  
Enterprise, 248  
Environment, 248  
Error prevention analysis, 248  
Establish SLA and contract road map, 164  
Evaluation, 249  
Evaluation and audit methods, 49  
Evaluation method, 48, 249  
Evaluation methods, major steps, 49  
generating a graphical representation of the results, 50  
identification, 48  
initial contact with the organization to be evaluated, 50  
on-site visits, 50  
summary of planning meetings results, 50  
Evaluation module, 249  
Evaluation process and support tools, 203  
Evaluation records, 249  
Evaluation report, 249  
Evaluation requester, 249  
evaluation results, 206  
Evaluation technology, 249  
Evaluation tool, 249  
Evaluation types, 51  
audit, 51  
external evaluation, 51  
joint evaluation, 51  
self-evaluation or internal evaluation, 51

- Evaluator, 249
- Event, 249
- Event/request management domain, 95
  - KPA interactions, 96
- Event/Request Management KPA, 97, 150
  - detailed exemplary practices, 150
    - Level 0, 150
    - Level 1, 150
    - Level 2, 150
  - expected results from, 98
  - goals of, 97
  - links with other KPAs, 98
- Event-driven basis, 249
- Event-driven review/activity, 249
- Event-tree analysis, 249
- Evolution and Correction Services KPA,
  - 112
    - expected results of, 113
    - goals of, 112
    - links to other KPAs, 113
- Evolution engineering domain, 107
  - KPA interactions, 108
  - KPAs, 107
  - support for, 117
- Evolution/correction road map, 179
- Execute services as per SLA and contract
  - road map, 165
- Executive, 249
- Exemplary practices
  - event/request management domain, 149
  - evolution engineering domain, 169
  - process management, 129
  - support to Evolution Domain, 187
- Exit criteria, 249
- External attribute, 249
- External measure, 249
- External quality, 250
- External service agreement, 32
- Facilitator, 250
- Facility, 250
- Failure, 250
- Failure mode and effects analysis (FMEA),
  - 250
- Fault, 250
- Fault isolation, 250
- Fault-tree analysis, 250
- Final transition preparation (products,
  - environment, and problem log) road map, 174
- Findings, 250
- Findings road map, 134
- Firmware, 250
- First-line manager, 250
- First-line support, 250
- Follow-up of planned and approved
  - activities road map, 159
- Follow-up on corrections and adjustments
  - road map, 193
- Formal method, 250
- Formal procedure, 250
- Formal review, 251
- Formalization, 251
- Framework quagmire, 63
- Frequency, 251
- FSM method, 251
- Full-time equivalent (FTE), 251
- Function, 251
  - Functional analysis, 251
  - Functional architecture, 251
  - Functional baseline, 251
  - Functional domain, 251
  - Functional requirement, 251
  - Functional size, 251
  - Functional user requirements, 251
- Generic practice, 252
- Goals, 252
- Grade, 252
- Group, 252
- Guideline, 252
- Hazard, 252
- Host computer, 252
- Human/user centered, 252
- Identification of baseline road map, 146
- Identify and analyze causes road map, 196
- Identify and document nonconformance
  - road map, 192
- Impact analaysis, 252
- Impact analysis (PR and MR) road map,
  - 158
- Impact analysis of a request for
  - modification road map, 156
- Implementation, 252
- Implementation model, 252

- Implemented process, 252
- Implied needs, 252
- Improvement, 252
- Incident, 252
- Indicator, 252
- Indirect measure, 252
- Indirect measurement method, 253
- Informal review, 253
- Information-gathering road map, 132
- Infrastructure, 253
- Initial level, 253
- Inspection, 253
- Installation (move to production of a change or version) road map, 185
- Institutionalization, 253
- Instrumentation, 253
- Integrated database, 253
- Integrated product development (IPD), 253
- Integrated product team (IPT), 253
- Integrated software management, 253
- Integration, 254
- Integrator, 254
- Integrity (of data), 254
- Integrity assurance authority, 254
- Integrity level, 254
- Interested party, 254
- Interface partner, 254
- Intermediate software product, 254
- Internal attribute, 254
- Internal measure, 254
- Internal quality, 254
- Investigate defects and failures road map, 196
- Involvement and communication with the developer, the customer, and purchasing road map, 170
- ISO 12207 standard, 17, 20, 21
  - levels, 20
  - maintenance processes, proposed update to, 229
- ISO 14764 standard, 17, 21
  - software maintenance categories, 23
- ISO 15504 (SPICE) standard, 44, 204
  - design process, 54
  - model, 57
  - reference model, 60
- ISO 90003 standard, 48
- ISO 9001 standard, 46, 48
- ISO 9126 standard—software product quality model, 28
- ISO software engineering standards, 17
- IT Service CMM Model, 56
- Item, 254
- Key factors of functional user requirements, 254
- Key practice, 43
- Key practices, 254
- key process area, 44, 254
- Life cycle, 255
- Life-cycle model, 255
- Likelihood, 255
- Local, 255
- Loss, 255
- Maintainability plan, 255
- Maintainer, 255
- Maintenance, 255
- Maintenance enhancement, 255
- Maintenance Innovation and Deployment KPA, 91
  - expected results from, 92
  - goals of, 91
  - links with other KPAs, 92
- Maintenance innovation and deployment, 146
  - detailed exemplary practices, 146
    - Level 0, 146
    - Level 1, 147
    - Level 2, 147
- Maintenance issues, 224
- Maintenance measurement, 23
- Maintenance Measurement and Analysis KPA, 123
  - expected results of, 124
  - goals of, 124
  - links with other KPAs, 124
- Maintenance measurement and analysis, 193
  - detailed exemplary practices, 193
    - Level 0, 193
    - Level 1, 194
    - Level 2, 194
- Maintenance plan, 255
- Maintenance planning (1 to 3 years) road map, 152

- Maintenance Planning KPA, 98, 151  
detailed exemplary practices, 151  
    Level 0, 151  
    Level 1, 151  
    Level 2, 152  
expected results from, 100  
goals of, 99  
links with other KPAs, 99
- Maintenance process, 255
- Maintenance process focus, 130  
detailed exemplary practices, 130  
    Level 0, 130  
    Level 1, 130  
    Level 2, 131
- Maintenance Process Focus KPA, 86  
expected results of, 87  
goals of, 86  
links with other KPAs, 86
- Maintenance process measurement, 23
- Maintenance process performance, 144  
detailed exemplary practices, 144  
    Level 0, 144  
    Level 1, 144  
    Level 2, 145
- Maintenance Process Performance KPA, 90  
expected results from, 91  
goals of this KPA, 90  
links with other KPAs, 90
- Maintenance process/service definition, 135  
detailed exemplary practices, 135  
    Level 0, 135  
    Level 1, 136  
    Level 2, 136
- Maintenance Process/Service Definition KPA, 87  
expected results of, 88  
goals of, 87  
links with other KPAs, 88
- Maintenance program, 255
- Maintenance review/acceptance process, 22
- Maintenance service contracts, 32
- Maintenance standards models, 227
- Maintenance training, 138  
detailed exemplary practices, 138  
    Level 0, 138  
    Level 1, 138  
    Level 2, 139
- Maintenance Training KPA, 88  
expected results from, 89
- goal of, 89  
links with other KPAs, 89
- Managed and controlled, 255
- Management, 256
- Management and control of the predelivery and transition process road map, 173
- Management of event and service request road map, 150
- Management practice, 256
- Management system, 256
- Manager, 256
- Mandatory provision, 256
- Marginal effectiveness, 256
- Marginal value, 256
- Maturity level, 256
- Maturity model validation, 57
- Maturity models, 41  
basic concepts, 42  
design, 52  
initial validation, 55  
inventory, 62  
maturity, 42  
process, 42  
typical architecture, 57
- Maturity questionnaire, 256
- Measurably significant effectiveness, 256
- Measurably significant value, 256
- Measure, 256
- Measurement, 256
- Measurement process, 256
- Mechanism, 256
- Method, 257
- Methodology, 257
- Migration, 257
- Migration processes, 22
- Model, 257
- Modification implementation process, 22
- Modification request (MR), 257
- Monitor, 257
- Monitoring, 257
- Multidisciplinary teamwork, 257
- Need, 257
- Nonconformity, 257
- Nondeliverable item, 257
- Nondevelopmental item, 257
- Nontechnical requirements, 257
- Normative, 257

- Objective evaluation road map, 192
- Objective evidence, 257
- Offered product, 258
- Offeror, 258
- Off-the-shelf product, 258
- Operational scenario, 258
- Operational software, 258
- Operational support, 258
- Operational Support Services KPA, 111, 175
  - detailed exemplary practices, 175
    - Level 0, 175
    - Level 1, 175
    - Level 2, 176
  - expected results of, 111
  - goals of, 111
  - links with other KPAs, 111
- Operations product, 258
- Operator, 258
- Opinion leader , 258
- Optimal effectiveness, 258
- Optimal value, 258
- Optimizing level, 258
- Optional, 258
- Optional provision, 258
- Organization, 258
  - Organization process maturity, 258
  - Organization's measurement program, 259
  - Organization's software process assets, 259
  - Organization's software process database, 259
  - Organization's standard software process, 259
  - Organizational process, 258
  - Organizational structure, 259
  - Organizational unit, 259
  - organizational unit is responsible for software maintenance, 15
- Orientation, 259
- Outsourcing, 259
  - Outsourcing Agreements, 34
- Package documentation, 259
- Pareto analysis, 259
- Participation in systems and acceptance tests road map, 175
- Partition, 260
- Peer review, 260
- Peer review leader, 260
- Perfective maintenance, 260
- Performance, 260
- Performance criteria, 260
- Performance requirement, 260
- Performed process, 260
- Periodic review, 260
- Periodic review/activity, 260
- Personal training road map, 142
- Plan, 260
- Planning and control measure, 260
- Planning disaster recovery testing road map, 156
- Policy, 261
- Practice, 261
- Practitioner, 261
- Predelivery and Transition Services KPA, 109
  - expected results of, 110
  - goals of, 109
  - links with other KPAs, 110
- Predelivery and Transition to Software Maintenance KPA, 170
  - detailed exemplary practices, 170
    - Level 0, 170
    - Level 1, 170
    - Level 2, 170
- Preventive action, 261
- Prime contractor, 261
- Probability, 261
- Problem and modification analysis process, 22
- Problem report, 261
- Procedure, 261
- Process, 261
  - assessment, 261
  - asset library, 261
  - assets, 261
  - attribute, 261
  - attribute rating, 262
  - capability, 43, 262
  - capability baseline, 262
  - capability determination, 262
  - capability determination sponsor, 262
  - capability level, 262
  - capability level rating, 262
  - category, 262
  - context, 262
  - database, 262
  - description, 262

- Process (*continued*)  
 development, 262  
 dimension, 262  
 domains, 76, 77, 262  
 effectiveness, 263  
 group, 263  
 improvement, 263  
 improvement action, 263  
 improvement program, 263  
 improvement project, 263  
 management, 263
- Process management context, 84
- Process management domain, 83  
 key process areas, 83  
 KPA interactions, 85
- Maintenance Innovation and Deployment KPA, 85  
 maintenance process focus, 84
- Maintenance Process Performance KPA, 85  
 Maintenance Training KPA, 84
- Process maturity, 43, 263
- Process measurement, 263
- Process measures, 263
- Process models, history of, 64
- Process outcome, 263
- Process performance, 43, 263
- Process performance baseline, 263
- Process profile, 264
- Process purpose, 264
- Process tailoring, 264
- Process, service, and software quality assurance, 190  
 detailed exemplary practices, 190  
 Level 0, 191  
 Level 1, 191  
 Level 2, 191
- Process, Service, and Software Quality Assurance KPA, 122  
 expected results of, 123  
 goals of, 122  
 links with other KPAs, 123
- Producibility, 264
- Product, 264  
 Product description, 264  
 Product liability, 264  
 Product measures, 264  
 Production deviation permit, 264  
 Production software monitoring road map, 176
- Profile, 264
- Program, 264  
 Program manager, 264
- Project, 264  
 Project manager, 265
- Project office, 265  
 Project representatives, 265  
 Project team, 265  
 Project's defined process, 265
- Promotion, 265
- Proposed capability, 265  
 Proposed change, 265
- Prototype, 265
- Qualification, 265  
 Qualification requirement, 265  
 Qualification testing, 266
- Qualify, 266
- Quality, 266  
 assurance, 266  
 audit, 266  
 characteristic, 266  
 control, 266  
 evaluation, 266  
 function deployment, 266  
 goals, 266  
 improvement, 266  
 in use, 266  
 loop, 266  
 management, 267  
 management system, 267  
 manual, 267  
 model, 267  
 objective, 267  
 plan, 267  
 planning, 267  
 policy, 267  
 requirement, 267  
 surveillance, 267  
 system, 267  
 system review, 267
- Quantitative control, 267  
 Quantitative level, 267  
 Quantitative management road map, 146
- Random error, 267
- Rating, 267  
 Rating level, 267
- Record, 268  
 Redocumentation, 268

- Redocumentation of product road map, 198
- Reengineering, 268
- Reengineering the product road map, 199
- Refactoring, 268
- Reference model, 268
- Reference user requirements (RUR), 268
- Regrade, 268
- Regression testing, 268
- Rejuvenation, 268
- Release, 268
- Reliability, 268
- Repair, 268
- Repeatable level, 268
- Repeatable process, 268
- Report, explain, and bill services road map, 167
- Request for proposal, 268
- Request management system, 269
- Request/Software Monitoring and Control
  - KPA, 100
    - expected results from, 101
    - goals of, 100
    - links with other KPAs, 101
  - Requests/Software Monitoring and Control
    - KPA, 159
      - detailed exemplary practices, 159
        - Level 0, 159
        - Level 1, 159
        - Level 2, 159
  - Required training, 269
  - Requirement, 269
  - Requirements analysis, 269
  - Requirements document, 269
  - Requirements traceability, 269
  - Requirements, plans, and resources road map, 141
  - Research of innovation/improvement road map, 147
  - Reservation, follow-up, and control road map, 190
  - Residual risk, 269
  - Resourcing, resourced, 269
  - Responsibility, 269
  - Responsibility and communication road map, 132
  - Restructuring, 269
  - Restructuring the product road map, 198
  - Retirement, 269
  - Return on investment (ROI), 269
  - Reverse engineering, 270
  - Reverse engineering the product road map, 198
  - Review, 270
  - Review and analyze progress road map, 161
  - Review data, 270
  - Review efficiency, 270
  - Reviews road map, 184
  - Rework, 270
  - Risk, 270
    - acceptance, 270
    - analysis, 270
    - assessment, 270
    - avoidance, 270
    - control, 270
    - identification, 270
    - management, 270
  - Role, 270
  - S3<sup>Assess</sup> evaluation support tool, 205
  - S3<sup>m®</sup> model, verification and validation of, 206
  - S3<sup>m®</sup> Process Model, 69
    - case studies, 210
    - context of software maintenance, 71
    - contributions to software product quality assessment during predelivery and transition, 211
    - contributions to the definition of software maintenance, 210
    - contributions to the definition of the service level agreement (SLA), 210
    - contributions to the improvement of a very small maintenance function, 213
    - assessment procedure, 214
    - assessment results, 215
    - maturity Level 0 practices assessment, 215
    - maturity Levels 1 and 2 practices assessment, 217
    - key process areas, 79
    - maturity profiles for levels 1 and 2, 209
    - perception of the usefulness of, 208
    - standards topics, 78
  - Safety, 271
  - SCAMPI, 51
  - SCI types, 271
  - Scrap, 271
  - Second-level support, 271
  - Security, 271

- Self-appraisal, 271  
 Senior manager, 271  
 Sensitivity analysis, 271  
 Service, 271  
 Service call, 271  
 Service level agreement (SLA), 30, 271  
     attitudes toward, 31  
     internal, 30  
 Service measurement, 29  
 Significant effectiveness, 271  
 Significant value, 271  
 Simulation, 271  
**SLA and Supplier Agreement Management**  
     KPA, 101, 162  
     detailed exemplary practices, 162  
         Level 0, 162  
         Level 1, 162  
         Level 2, 163  
     expected results from, 102, 162  
     goals of, 102, 162  
     links with other KPAs, 102, 162  
 Small enhancement, 272  
 Software, 272  
**Software acquisition management**  
     personnel, 272  
 Software acquisition plans, 272  
 Software acquisition process, 272  
 Software acquisition process assets, 272  
 Software acquisition process group, 272  
 Software acquisition process repository, 272  
 Software acquisition process-related documentation, 273  
 Software acquisition project, 273  
 Software architecture, 273  
 Software baseline audit, 273  
 Software baseline library, 273  
 Software build, 273  
 Software capability evaluation, 273  
 Software configuration control board, 273  
 Software configuration item (SCI), 273  
 Software configuration management (SCM), 273  
 Software configuration management plan (SCM plan), 273  
 Software development plan, 273  
 Software development process, 1  
**Software Engineering Body of Knowledge (see SWEBOK)**
- Software engineering environment (SEE), 273  
 Software engineering group, 274  
 Software engineering personnel, 274  
 Software engineering staff, 274  
**Software Evolution and Correction Services**  
     KPA, 178  
     detailed exemplary practices, 178  
         Level 0, 178  
         Level 1, 178  
         Level 2, 179  
 Software integration, 274  
 Software integrity level, 274  
 Software library, 274  
 Software life cycle, 274  
 Software maintainers' context diagram, 71  
 Software maintenance activities, 14  
 Software maintenance as a primary process of ISO/IEC 12207, 16  
 Software maintenance benchmarking, 35  
 Software maintenance categories, 23  
 Software maintenance issues, 1  
     maintainers' perceptions, 4  
     users' perceptions, 2  
 Software maintenance key processes, 22  
**Software maintenance measurement**  
     program, 26  
     estimation, 27  
     experience, 27  
**Software maintenance operational processes**, 75  
**Software maintenance organizational processes**, 76  
**Software maintenance process and activities**, 21  
     characteristics, 25  
 Software maintenance SLA, 32  
 Software maintenance standards, 15, 227  
 Software maintenance support processes, 76  
 Software maintenance, definition, 10  
     initial development, 10  
     maintenance and use, 10  
**Software maintenance, key process areas**, 76  
 Software maintenance, ontology, 24  
 Software management approaches, 1  
 Software manager, 274  
 Software maturity models, inventory of, 65  
 Software measurement perspectives, 24

- Software migration road map, 199  
Software plans, 274  
Software process, 43  
    assessment, 274  
    description, 275  
    element, 275  
    improvement plan, 275  
    improvement proposal, 275  
    maturity, 275  
Software product, 275  
    developer, 275  
    evaluation, 275  
    measurement, 28  
Software project, 275  
Software quality assurance, 276  
Software quality goal, 276  
Software quality management, 276  
Software Rejuvenation, Migration, and Retirement KPA, 125, 197  
    detailed exemplary practices, 197  
        Level 0, 197  
        Level 1, 197  
        Level 2, 198  
    expected results of, 126  
    goals of, 126  
    links with other KPAs, 126  
Software requirement, 276  
Software retirement road map, 200  
Software service, 276  
Software support, 276  
Software test environment (STE), 276  
Software tool, 276  
Software transition, 276  
Software transition planning road map, 154  
Software unit, 276  
Software work product, 276  
Software-acquisition-related group, 273  
Software-process-related documentation, 275  
Software-related contractual requirements, 276  
Software-related group, 276  
Solicitation package, 277  
Special cause (of a defect), 277  
Special causes of variation, 277  
Specific practice, 277  
Specification, 277  
Sponsor, 277  
Staff, 277  
Stage, 277  
Staged architecture, 277  
Stakeholder, 277  
Stakeholder requirement, 277  
Standard, 278  
Standard process, 278  
Standard software acquisition process, 278  
Standards pyramid, 19  
State, 278  
Statistical quality control, 278  
Subcontractor, 278  
Submitter, 278  
Subsystem, 278  
Supplier, 278  
Supporting function, 278  
Sustainability, 278  
SWEBOK (Software Engineering Body of Knowledge), 8  
    categories of maintenance, 8  
    fundamentals, 8  
    maintenance issues, 9  
    software maintenance process, 9  
    techniques for software maintenance, 9  
Synthesis, 278  
System, 278  
    architecture, 278  
    component, 278  
    design process, 279  
    engineering group, 279  
    integrity level, 279  
    requirement, 279  
System requirements allocated to software requirements, 279  
System technical requirement, 279  
Systematic error, 279  
Systematic failure, 279  
Systems engineering, 279  
Systems engineering process group (SEPG), 279  
Tailor, 279  
Tailored process, 279  
Tailoring, 279  
Target capability, 279  
Target computer, 280  
Task, 280  
Task kick-off meeting, 280  
Task leader, 280  
Team, 280

- Technical objectives, 280
- Technical performance measurement (TPM), 280
- Technical requirements, 281
- Technical software requirements, 281
- Techniques, 281
- Technology, 281
- Term assignments for students, 231
- Test, 281
  - Test case, 281
  - Test coverage, 281
  - Testability, 281
  - Testing (unit, integration, and regression) road map, 181
- Third-level support, 281
- Threat, 281
- Ticket, 281
- Tool, 281
- Top management, 282
- Total quality management (TQM), 282
- Traceability, 282
- Trade study, 282
- Trade-off, 282
- Train, 282
- Training, 282
- Training and knowledge transfer control road map, 173
- Training for projects in predelivery and transition road map, 143
- Training group, 282
- Training of new personnel road map, 142
- Training program, 282
- Training waiver, 282
- Transition, 282
- Transitioned, 282
- Trillium design process, 52
- Trillium maturity levels, 54
- Unit, 282
- Urgent changes and corrective measures road map, 161
- User, 282
- User documentation, 283
- User requirements baseline, 283
- User support, 283
- User-training road map, 143
- Validation, 283
- Value, 283
- Variation, 283
- Verification, 283
- Verification and Validation KPA, 113, 182
  - detailed exemplary practices, 182
    - Level 0, 182
    - Level 1, 182
    - Level 2, 183
  - expected results of, 114
  - goals of, 114
  - links with other KPAs, 114
- Verify, 283
- Version, 283
- Version/release and upgrade planning road map, 156
- Waiver, 283
- Well-defined process, 283
- Work breakdown structure (WBS), 283
- Work environment, 283
- Work product, 284
- Written procedure, 284

# About the Authors



Alain April is professor at the École de Technologie Supérieure, Université du Québec, Montréal, Canada. He obtained his Ph.D. in Software Engineering from the Otto von Guericke University in Magdeburg, Germany, in software engineering. He has worked in the banking and telecommunications industries for more than 20 years. Professor April was co-editor of Software Quality International Standard ISO 9126 (part 3). He is the co-editor of the quality and software maintenance chapters of the *Guide to the SWEBOk (Software Engineering Body of Knowledge)*.



Dr. Alain Abran is a professor and the director of the Software Engineering Research Laboratory at the École de Technologie Supérieure (ETS)—Université du

Québec ([www.gelog.etsmtl.ca](http://www.gelog.etsmtl.ca)). He was the co-executive editor of the *Guide to the Software Engineering Body of Knowledge* ([www.swebok.org](http://www.swebok.org)). From 2001 to 2003, he was also actively involved with software engineering standards as the international secretary for ISO/IEC JTC1 SC7—Software and System Engineering, and is now co-chair of the Common Software Measurement International Consortium (COSMIC). Dr. Abran has more than 20 years of industry experience in information systems development and software engineering. He holds a Ph.D. in Electrical and Computer Engineering (1994) from the École Polytechnique de Montréal (Canada) and Master degrees in Management Sciences (1974) and Electrical Engineering (1975) from the University of Ottawa. His research interests include software productivity and estimation models, software engineering foundations, software quality, software functional size measurement, software risk management, and software maintenance management.