

The image shows the front cover of a book titled "IT Project Management Handbook". The cover is dark blue with a pattern of binary code (0s and 1s) in a lighter shade of blue. The title "IT Project Management Handbook" is written in white, with "IT" in a large, bold font. Below the title, the authors' names "Jae Nyeon Kim" and "Prince Saha" are listed. At the bottom, the publisher's name "BY PEARSON EDUCATION" is visible.

IT

Project
Management
Handbook

Jae Nyeon
Kim, Prince Saha

BY
PEARSON EDUCATION

Table of Contents

IT Project Management Handbook.....	1
Introduction.....	3
Section I: Information Technology Process.....	6
Chapter List.....	6
Chapter 1: Information Technology Project Initiation.....	7
Effective IT Project Management Techniques.....	7
IT Project Management Objectives.....	9
Characteristics of a Good IT Project Manager.....	9
Personal Characteristics.....	9
Project–Related Characteristics.....	10
Team–Related Characteristics.....	10
IT Project Initiation.....	10
Identification of Project External Interfaces.....	12
Identification of Users, Customers, and Stakeholders.....	12
Users.....	13
Customers.....	13
Stakeholders.....	13
IT Project Development Life Cycle Phases.....	14
Case Study 1–1.....	15
Project Management Checklist.....	16
Chapter 2: Guidelines for Successful Project Planning.....	18
IT Project Planning.....	18
Project Book.....	18
Establishing Project Objectives.....	19
Setting Project Goals.....	19
Reporting Schema.....	20
Planning Project Recovery.....	20
Project Modeling and Simulation.....	20
Work Breakdown Structure.....	22
Scheduling the Work Breakdown Structure.....	22
Milestone Model.....	23
Budgeting.....	23
Project Organization.....	24
Functional.....	25
Pure.....	26
Matrix.....	26
Hybrid.....	27
Staffing.....	28
Revisiting Case Study 1–1.....	30
Project Planning Checklist.....	32
Chapter 3: Project Estimating Techniques and Tools.....	33
Overview.....	33
Cost–Estimating Tools.....	33
Constructive Cost Model.....	33
COCOMO Example.....	36

Table of Contents

Chapter 3: Project Estimating Techniques and Tools	
COCOMO II.....	36
Function Points Method.....	37
COCOMO and the Function Points Method.....	38
Earned Value Tool.....	39
Use of Earned Value Data.....	41
Other Cost Estimation Tools.....	41
Scheduling Techniques.....	41
Gantt Chart.....	42
PERT Chart.....	43
Critical Path Method.....	47
Case Study 3–1.....	47
Chapter 4: Project Management Quality Control Mechanism.....	50
System Quality Management.....	50
System Quality Characteristics.....	50
Quality Metrics.....	51
Quality Indicators.....	51
Quality Parameters.....	52
Technical Reviews.....	53
Technical Walk–Through.....	53
System Quality Improvement Process.....	54
System Software Quality and Interoperability.....	54
Software Development Capability Maturity Model.....	56
QAS Checklist.....	58
Section II: Industry Best Practices.....	60
Chapter List.....	60
Chapter 5: IT Project Risk Management.....	61
What is Project Risk?.....	61
Risk Factors.....	61
Risk Management.....	63
Private Risk Management Plan.....	63
Risk Assessment Plan.....	64
Risk Mitigation Plan.....	65
Risk Monitoring and Reporting Plan.....	67
Principal Best Practices.....	67
Formal Risk Management.....	68
Agreement on Interfaces.....	68
Formal Inspection.....	68
Metric–Based Scheduling and Management.....	68
Binary Quality Gates at the Inch–Pebble Level.....	69
Project–Wide Visibility of Progress Versus Plan.....	69
Defect Tracking Against Quality Targets.....	69
Configuration Management.....	70
People–Aware Management Accountability.....	70
Risk Management Checklist.....	70

Table of Contents

Chapter 6: Commercial Best Practices Standards.....	73
Commercial System Development Standards.....	73
Characteristics of a Good System Development Standard.....	73
System Management Standard.....	74
IEEE Standards.....	74
Standard for Application and Management of the Systems Engineering Process.....	75
Standard for IT: Software Life–Cycle Processes.....	76
ISO 9000 Standards.....	79
Tailoring Standards Techniques.....	80
Guidelines for Tailoring.....	81
IT Project Standards Checklist.....	81
Chapter 7: System Measurement Techniques.....	88
Project Measurement.....	88
Project Measurement Metrics.....	89
Case Study 7–1.....	90
Management Measurement Criteria.....	91
Best Practices System Metrics.....	92
Case Study 7–2.....	93
Project Software Measurement.....	94
Tailoring Project Measurement.....	95
Practical Software Measurement.....	95
Project Measurement Checklist.....	96
Chapter 8: Commercial Items.....	98
Definition of CI Tools.....	98
CI Selection Criteria.....	98
Ease of Use.....	99
Capability.....	99
Robustness.....	99
Functionality.....	100
Ease of Insertion.....	100
Quality of Support.....	100
Microsoft Project 98.....	101
Kidasa Software's Milestones Etc. 5.0.....	102
AEC Software's FastTrack Schedule 6.0.....	104
Digital Tools, Inc., AutoPlan.....	105
Computer Associates International, CA–SuperProject for PC.....	106
COSMIC, COMPASS.....	107
Quality Software Products Co., MasterPlan.....	108
Cambridge Management Systems, Project Outlook.....	109
Risk Radar.....	109
Commercial Items Checklist.....	111
Section III: Using the Latest Technology.....	112
Chapter List.....	112
Chapter 9: Customer, Customer, and Customer.....	113
Knowing the Customer.....	113
Customer Relationship Management Process.....	114

Table of Contents

Chapter 9: Customer, Customer, and Customer	
Customer Expectations.....	114
Customer Acceptance Criteria.....	116
Case Study 9–1.....	117
Customer Equation.....	117
Manpower and Personnel Integration.....	117
Human Factor Engineering.....	118
Human System Integration.....	118
System Safety.....	119
User–Machine Interfaces.....	120
System Security.....	120
Case Study 9–2.....	122
Customer Management Checklist.....	122
Chapter 10: Network Management.....	123
Telecommunications Basics.....	123
Modems.....	123
IT Networks.....	124
LAN.....	124
WAN.....	124
Internet Technology.....	125
Wireless Technology.....	125
Challenges to the Project Manager.....	125
Wireless Data Providers.....	126
Wireless Devices and Platforms.....	127
Uses of Wireless Technology.....	128
Future of Wireless Technology.....	128
Cellular Communications.....	129
Virtual Private Networking.....	129
Chapter 11: Internet Applications.....	130
Emerging Wireless Technologies Standard.....	130
Wireless Applications.....	131
E–Commerce and M–Commerce.....	131
Wireless Communication.....	132
Extensible Markup Language.....	133
BlueTooth.....	134
Handheld Device Markup Language.....	134
Information Architecture.....	134
Chapter 12: Distributed Object Technology.....	135
Distributed Object Environment.....	135
Distributed Computing Environment.....	135
Common Object Request Broker Architecture.....	136
CORBA Benefits.....	137
Interface Definition Language.....	137
CORBA Architecture.....	138
Common Object Model.....	138
Distributed COM.....	138
ActiveX.....	139

Table of Contents

Chapter 12: Distributed Object Technology	
CORBA 2.0 and Interoperability.....	139
Distributed System Object Model.....	140
CORBA 3.0.....	140
Case Study 12–1.....	140
Case Study 12–2: Management of a Bank Account.....	141
Case Study 12–3: A–Bank Information System.....	142
Chapter 13: Distributed Objects.....	144
Case Study 13–1: An Airline Reservation System (OnTime Airline).....	144
Customer Requirements and Constraints.....	144
Distributed Object Technology.....	144
Two– and Three–Tier Client–Server Environments.....	145
Service Objects.....	145
Service Object Replication.....	146
Partitioning.....	146
Load Balancing.....	146
Failover.....	147
Use Cases.....	147
Use Case General Information.....	147
Use Case Analysis Information.....	147
Use Case Design Information.....	147
Discussion.....	149
Chapter 14: Wireless Practical Case Study.....	151
Continuation of Case Study 13–1.....	151
Requirements Analysis.....	151
Design.....	151
Development of a Logical Model for the Application.....	152
Definition of Acceptable User Response Time When Submitting a Request to the System.....	152
Operational Requirements.....	152
Backup Plan.....	153
Definition of Practitioners Roles.....	153
Technical Architect Roles.....	153
Information Architect Roles.....	154
Life–Cycle Phases.....	154
Development Environment.....	154
Testing Environment.....	155
Integration Environment.....	155
Staging Environment.....	155
Production Environment.....	155
Section IV: Building an IT System.....	156
Chapter List.....	156
Chapter 15: System Requirements.....	157
System Requirement Identification.....	157
Requirement Determination.....	157
Requirement Elicitation.....	158
Requirement Definition.....	159

Table of Contents

Chapter 15: System Requirements	
Importance of a Good Requirement.....	159
Samples of Good, Bad, and Ugly Requirements.....	159
Good Requirements.....	159
Bad Requirements.....	160
Ugly Requirements.....	160
Requirement Types.....	160
Characteristics of a Good Requirement Specification.....	161
Unambiguous.....	161
Complete.....	161
Verifiable.....	162
Consistent.....	162
Correct.....	162
Modifiable.....	162
Traceable.....	162
Testable.....	162
Usable after Development.....	163
System Requirement Specification.....	163
Preparation of Test Plans, Procedures, and Test Cases.....	166
Case Study 15–1.....	166
System Requirement Checklist.....	168
Chapter 16: Managing System Requirements.....	169
Requirements Management Process.....	169
Requirements Management Tools.....	170
Selection Criteria of a Good Requirements Management Tool.....	171
TBI Caliber–RM Tool.....	172
Requirements–Driven Development and Testing.....	172
Clear Communication.....	173
Shared Requirements.....	173
Internet–Based Nature.....	173
Traceability Throughout the Life Cycle.....	174
Change Management of Requirements.....	175
External References.....	175
Automatic Requirement Import.....	175
Comprehensive Reporting.....	176
Integration of the Life Cycle.....	176
Flexibility to Support Processes.....	177
Applications Support.....	177
E–Commerce.....	177
Enterprise Resource Planning.....	177
Supply Chain Management.....	178
Chapter 17: System Design Process.....	179
System Design.....	179
System Design Characteristics.....	180
Configuration Item Selection Criteria.....	180
Allocation of Requirements to CSCI.....	180
IT System Architectural Model.....	181
Exploration of Reuse Assets.....	182

Table of Contents

Chapter 17: System Design Process	
Establishment of a System Reuse Plan.....	182
SRP Purpose.....	183
Various Responsibilities and Duties.....	183
Operation and Maintenance of the Reuse Center.....	184
Customer and User Participation.....	184
Configuration Management.....	184
Quality Assurance and Control.....	184
Training.....	184
Repository System Establishment and Maintenance.....	185
Reuse Life–Cycle Phases.....	185
Case Study 17–1.....	185
Selection Criteria of a Good Database Management System.....	186
Performance.....	186
Ease of Use.....	186
Migration.....	187
Data Modeling Capabilities.....	187
Data Dictionary.....	189
System Design Document.....	189
Operational Concepts.....	189
System Architecture.....	189
System Design Features.....	189
Processing Resources.....	190
Requirements Traceability.....	190
System Design Checklist.....	190
Chapter 18: Software Requirements.....	192
Software Requirements Process.....	192
Software Development Plan.....	193
Software Life–Cycle Model.....	193
Grand Design Model.....	194
Incremental Model.....	194
Evolutionary Model.....	195
Waterfall Model.....	195
Prototype and Simulation Model.....	196
Assembling Reusable Components Model.....	197
Spiral Model.....	197
Operational Model.....	198
Transformational Model.....	199
Star Process Model.....	199
Domain Prototype Model.....	199
Domain Functional Model.....	200
Case Study 18–1.....	201
Software Requirement Specification.....	202
Chapter 19: Software Design Process.....	204
Software Design Architecture.....	204
What is Software Design?.....	204
Software Design Methods.....	205
Object–Oriented Method.....	205

Table of Contents

Chapter 19: Software Design Process	
Object Attribute.....	207
Object Behavior Model.....	210
Structured Design Method.....	212
Software Design Description.....	213
Chapter 20: Software Implementation Process.....	216
Overview.....	216
Software Implementation.....	216
Software Implementation with Reuse.....	216
New Development.....	217
Reverse Engineering.....	217
Reengineering.....	217
Exploration of the Internet for Suitable Reusable Assets.....	218
Case Study 20–1.....	218
Software Testing.....	219
Software Testing Best Practices.....	220
Static Analysis Techniques.....	220
Dynamic Analysis Techniques.....	220
Case Study 20–2.....	221
Software Integration Testing.....	222
Configuration Management.....	223
Software Change Process.....	224
Preparation of the Engineering Change Proposal.....	224
Configuration Control Board.....	224
Section V: IT Project Completion and Evaluation.....	226
Chapter List.....	226
Chapter 21: System Integration and Evaluation.....	227
System Integration Process.....	227
System Integration Testing.....	227
Case Study 21–1: Doomed Mars Spacecraft.....	229
System Requirements Verification and Validation.....	230
Management of IT Project Documentation.....	231
Customer Acceptance Testing.....	231
Management of the IT Project Audit.....	232
Well–Managed, Successful IT Project.....	232
Unsuccessful IT Project.....	234
Future Trend.....	236
Checklist: Suggestions for a Successful IT System Project.....	237
Chapter 22: Practical Case Study.....	239
Company Background.....	239
Corporate MIS Structure.....	239
Conflict.....	239
Hardware and Network Configuration.....	240
Software Environment.....	240
Corporate Service Professionals.....	240
Role of Business Services and Technical Support.....	241

Table of Contents

Chapter 22: Practical Case Study

Role of Software Development and Maintenance.....	241
Program.....	241
Program Execution.....	242
Results.....	243
Lessons Learned.....	243
Risk Analysis.....	244
Acronyms.....	244
A–C.....	244
D.....	246
E–I.....	247
L–O.....	248
P–S.....	249
T–X.....	252
References.....	253
List of Figures.....	258
Chapter 1: Information Technology Project Initiation.....	258
Chapter 2: Guidelines for Successful Project Planning.....	258
Chapter 3: Project Estimating Techniques and Tools.....	258
Chapter 4: Project Management Quality Control Mechanism.....	258
Chapter 5: IT Project Risk Management.....	259
Chapter 7: System Measurement Techniques.....	259
Chapter 8: Commercial Items.....	259
Chapter 9: Customer, Customer, and Customer.....	259
Chapter 11: Internet Applications.....	259
Chapter 12: Distributed Object Technology.....	259
Chapter 13: Distributed Objects.....	260
Chapter 14: Wireless Practical Case Study.....	260
Chapter 15: System Requirements.....	260
Chapter 16: Managing System Requirements.....	260
Chapter 17: System Design Process.....	260
Chapter 18: Software Requirements.....	260
Chapter 19: Software Design Process.....	261
Chapter 20: Software Implementation Process.....	261
Chapter 21: System Integration and Evaluation.....	261
Chapter 22: Practical Case Study.....	262
List of Tables.....	263
Chapter 1: Information Technology Project Initiation.....	263
Chapter 2: Guidelines for Successful Project Planning.....	263
Chapter 3: Project Estimating Techniques and Tools.....	263
Chapter 5: IT Project Risk Management.....	263
Chapter 9: Customer, Customer, and Customer.....	263
Chapter 13: Distributed Objects.....	263
List of Boxes.....	264
Chapter 1: Information Technology Project Initiation.....	264
Chapter 6: Commercial Best Practices Standards.....	264

Table of Contents

List of Boxes

Chapter 9: Customer, Customer, and Customer.....	264
Chapter 15: System Requirements.....	264

IT Project Management Handbook

Jag Sodhi Prince Sodhi

8230 Leesburg Pike, Suite 800
Vienna, VA 22182
Telephone: (703) 790-9595
Fax: (703) 790-1371
www.managementconcepts.com

Copyright © 2001 by Management Concepts

All rights reserved. No part of this book may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording, or by an information storage and retrieval system, without permission in writing from the publisher, except for brief quotations in review articles.

Printed in the United States of America

Library of Congress Cataloging-in-Publication Data

Sodhi, Jag.

IT project management handbook / Jag Sodhi, Prince Sodhi.

p. cm.

Includes bibliographical references and index.

ISBN 1-56726-098-5 (hc.)

1. Industrial project management. 2. Information technology-Management. I. Sodhi, Prince. II. Title.

HD69.P75 S625 2000

658.4'04-dc21

00-051107

01 02 03 04 9 8 7 6 5 4 3 2 1

We dedicate this book to welcome Armaan Sodhi into the family.

Acknowledgments

We would like to thank the editors and staff at Management Concepts, Inc., who convinced us of the need for this book and supported us in its development. We owe a great debt especially to Cathy Kreyche, Management Concepts, Inc., and Cara Moroze, Technology Builders, Inc.

We wholeheartedly thank all our friends, students, and colleagues who contributed to the completion of *IT Project Management Handbook* in different ways. We thank the staff at Management Concepts, Inc. who

IT Project Management Handbook

were involved in the editing and production of the text.

Finally, we are grateful to the members of our families who diligently supported us so that this project could be successfully finished. Without their love and support, we would not have completed such a challenge.

Jag Sodhi

Prince Sodhi

About the Authors

Jag Sodhi, MS, is a professor associated with the Defense Acquisition University. He has 32 years of experience as a well-known author, software developer, consultant, instructor, and project manager for defense products and commercial business applications. He has been a computer specialist and project manager and a member of DOD's advanced technology team. He specializes in the best commercial practices and COTS tools for systems development and maintenance.

Prince Sodhi is a project manager for Brience, Inc. He is an expert in project management, several programming languages, and distributed computing applications in multiple-tier, client-server environments. He is co-author with Jag Sodhi of the books *Software Reuse: Domain Analysis and Design Process* (1999), and *Object-Oriented Methods for Software Development* (1996), McGraw-Hill. He has a bachelor's degree in computer/mathematics from the University of Minnesota.

Introduction

Information technology (IT) projects are receiving great attention in the computer industry because they touch almost everyone's lives. Whether IT projects are managed for business, financial, academia, government, military, or nonprofit organizations, accurate computerized information is needed to make good decisions in less time. However, this computerized information is only as good as the design and management of the IT project systems.

Based on real-world practical experiences and case studies, *IT Project Management Handbook* presents a framework for managing, improving, and building greater accuracy into every phase of IT project management. The successful IT project produces the highest quality products with the fewest number of defects in the shortest, most cost-effective manner that satisfies customer needs. As a primary benefit, *IT Project Management Handbook* provides practical guidelines for managing computer projects—better, faster, and cheaper.

An IT project manager has to make decisions about managing the risks for cost overruns, schedule delays, and failures to meet customer requirements. A successful IT manager takes control of scheduling, budgets, resource allocation, and teamwork with specific strategies to help reach correct decisions. A good manager is an asset in any organization and applies proven project management techniques to avoid common pitfalls at every step of system development. *IT Project Management Handbook* provides the manager with best practices for managing an IT project, thus enabling the reader to evolve into a more effective IT project manager.

IT Project Management Handbook offers practical IT management strategies and well-tested methods and tools for implementing project management techniques. It evaluates recent advances in IT project management techniques, systems development, Internet technology, software repositories, COTS, and system software development technologies. The availability of system software components via the Internet has become one of the most discussed topics among practitioners for reducing costs, maintaining schedules, and producing quality software for successful IT projects. Economically reusing well-tested software components that are suitable to the needs and requirements of the project manager is a key factor presented in this text.

IT Project Management Handbook consists of five sections that discuss the different aspects of the IT project management process. Section I introduces the basics of project management. Chapter 1 explains effective management techniques, objectives, initiations, and characteristics of a good IT project manager, identifying external interfaces, users, customers, and stakeholders.

Chapter 2 provides the guidelines for successful project planning. Planning for an IT project requires the coordination and management of activities, phases, schedules, time lines, and staffing, as well as the delegation of duties and responsibilities. This chapter also discusses the adoption of a reuse approach to system development and maintenance throughout the life cycle of projects.

Chapter 3 discusses correctly estimating costs, defining the appropriate levels of efforts, WBS, scheduling, staffing, milestones, budgeting, and IT system development and maintenance phases. COCOMO predicts the effort and duration of a project. The *function points* method measures the system's development productivity. An *earned value* tool keeps a record of tasks and activities. The *Gantt chart* shows activities, their duration, and their interrelationships. The *PERT chart* identifies the relationship of many steps involved in complex systems. A *CPM tool* is designed to reduce the length of a project.

Chapter 4 introduces quality control mechanisms for IT project management. Establishing a system for developing efficient practices will monitor these practices throughout the life cycle of the project. IT project

Introduction

managers establish the QAS to control established quality principles, and they insist that quality is the responsibility of all who are involved in the system development, and not simply the responsibility of those in the quality assurance section.

Section II presents industry best practice standards. IT project managers establish the processes necessary to identify, monitor, and manage risks. Chapter 5 discusses managing IT project risk, which is often the result of unpredictable losses that can occur anytime and anywhere during the project's life cycle. Risk is associated with all aspects of the project—requirements, architecture, design, implementation, team members, management, and WBS.

Chapter 6 discusses commercial best–practice standards. Most system developers design their own standards and follow them in their organizations. This chapter studies IEEE standards for system development and management, ISO standards, and guidelines to tailor standards.

Chapter 7 explains the importance of system measurement techniques, methods, and tools. Measurement is a key element in the successful management of IT projects in every established engineering discipline. The purpose of this chapter is to provide project managers with the systems information that is needed to monitor and correct project progress, activities, costs, schedules, and technical objectives.

Chapter 8 discusses commercial items (CI). CIs consist of automated software tools for managing IT projects. CI tools also include COTS and NDI. The CI is not a replacement for any method of the system management; rather, it is a supplement for the methods and enhances the probability for generating quality products.

Section III introduces the latest technology for IT project management. Chapter 9 discusses the customer. The quality of the management of an IT system is determined not only by the absence of system defects but also by customer satisfaction. The customer is an important factor in the growth of every organization, and customer satisfaction plays a vital role in building an organization's reputation and progress. This chapter offers proven strategies to IT managers to ensure the building of long–term customer relationships.

Chapter 10 explains the basics of network management. Currently, IT project management requires networking to ensure that data and information are electronically accessed at a distance.

Chapter 11 discusses Internet applications and e–commerce. Since the existence of the Internet, the World Wide Web has evolved to the point that wireless technology now allows individuals to 'surf' the web.

Chapter 12 discusses distributed object technology, which provides the mechanisms for linking objects in a network. With distributed objects, users concentrate more on what is needed, not how to get it. This chapter also discusses environments and CORBA.

Chapter 13 presents a practical case study using distributed object technology. The discussion surrounding the case study focuses on three–tier client–server environments, service objects, service–object replication, partitioning, load balancing, and failover.

Chapter 14 discusses a wireless practical case study and building wireless components.

Section IV presents the requirements for building the foundations of IT systems. Failure to understand the requirements in the beginning may result in an inefficient system and delays in product delivery.

Chapter 15 discusses system requirements, which basically consist of hardware, software, and operational requirements. IT project managers establish processes that identify, elicit, and understand system requirements.

Introduction

Chapter 16 introduces managing system requirement processes, tools, and activities that span the life cycle of an IT system. These activities relate to requirement changes and implementation during system development and maintenance.

Chapter 17 discusses system design process. IT project managers establish system design process, and the system designer develops the system architecture to understand the requirements. System design begins with the baseline of system requirements. System requirements are partitioned between software and hardware requirements.

Chapter 18 discusses software requirement analysis, which is a process of understanding the requirements of the technical practitioners. IT project managers select the methods for software requirement analysis. This chapter discusses software development planning, various modeling techniques, and software requirement analysis.

Chapter 19 studies software design. The software design process translates requirement specifications from the 'what' phase of software development to the 'how' phase. This chapter covers software design architectures, software design methods, object-oriented design methods, structured design methods, and software design descriptions.

Chapter 20 discusses the software implementation process, which includes writing source code, executing code, and testing code for each item in the design phase, as well as integrating software units and software components into software items. This chapter also discusses testing software units and components to ensure satisfaction of requirements.

Section V includes IT project completion and evaluation. An effective IT manager successfully completes an IT project within budget and schedule. Chapter 21 discusses system integration and the evaluation process of combining hardware configuration items with software configuration items, as well as other systems as necessary, into the IT system. This chapter also discusses the verification of system requirements, validation, documentation, customer acceptance testing, future trends, and guideline suggestions.

Chapter 22 presents a practical case study from start to finish.

Acronyms used in this book are defined and references are provided at the end of the book.

The information and data contained in this book have been compiled from various sources and are intended to be used for reference purposes. Neither the publisher nor the authors guarantee the accuracy of the information and data.

Section I: Information Technology Process

Chapter List

Chapter 1: Information Technology Project Initiation

Chapter 2: Guidelines for Successful Project Planning

Chapter 3: Project Estimating Techniques and Tools

Chapter 4: Project Management Quality Control Mechanism

Accurate computerized information is as good as the design and management of the IT project systems.

Chapter 1: Information Technology Project Initiation

Information technology (IT) projects receive attention in the computer industry because they influence almost everyone. Whether the IT project manager is managing projects for business, financial, academic, government, military, or nonprofit organizations, he or she needs accurate computerized information to make decisions in a short amount of time. This computerized information is as good as the design and management of the IT project systems.

Effective IT Project Management Techniques

A modern IT project manager's task is threefold: to supervise IT computer professionals, understand state-of-the-art techniques, and make the IT project successful. The manager should enhance his or her ability to understand project management techniques, modern technologies, and system development methods and tools. A manager cannot effectively manage a technical team unless he or she understands the basics of what the team members are doing and the technical aspects of the organization. In this way, he or she increases the efficiency of the operation, maintains a positive environment, and reduces turnover.

A good IT project manager is hard to find. A manager must be able to handle his or her team well to deliver a quality product within the defined budget and schedule. A manager who is well versed in the state-of-the-art techniques can analyze the user's requirements, design the IT system, develop the software, and deliver the finished product to the satisfaction of the users.

A good manager must recognize what tools are required and use them in a knowledgeable manner to plan, estimate, schedule, and develop the IT project's complete work breakdown structure (WBS) without guesswork or reliance upon another individual's memory or experience. The manager is a leader who increases interpersonal relationships with his or her group, communicates effectively, and guides with the vision necessary to lead them to success. He or she is practical in making decisions and recognizes the importance of objectivity, vision, and initiative in arriving at sound, quality decisions.

A good manager also identifies and develops talent in others. He or she arranges special IT technical training to foster individual growth and efficiently manages time and resources through the delegation of tasks to those who are the most suitable. The manager is constantly assessing skills in others and escalating responsibilities accordingly. He or she gives due respect, promotions, and raises to maintain high morale of team members.

The manager selects the methodology and techniques that are the most suitable for development of the IT system. He or she staffs the project in phases as needed and provides the necessary technical training to bring himself or herself and the team up to standard. The manager understands users' acceptance criteria of the system products. He or she stresses that the team members should develop a prototype model and simulate the system to aid in understanding the users' requirements for the development and maintenance of the system. The manager encourages the users to examine cases at the completion of the project to confirm that the users' requirements are testable. The manager appoints other supporting personnel as shown in Box 1–1. Figure 1–1 graphically presents this system of support personnel and various phases

Box 1–1: List of Support Personnel

- Technical training consultant
- System engineers and analysts
- System testers

IT Project Management Objectives

The IT project manager must understand the users' requirements, hire quality team members, and manage the team members well. He or she must be able to complete the project successfully on time, within the budget, and to the satisfaction of the users.

System development has faced problems in the IT industry since its inception. The cost of hardware is declining with time, whereas the cost of software development is steadily increasing with time (Figure 1–2). Many IT projects are either challenged or not completed, and challenged projects are often completed with cost overrun and delays in delivery schedules. The percentage of application development failures is greater than the number of successfully completed projects for many reasons.

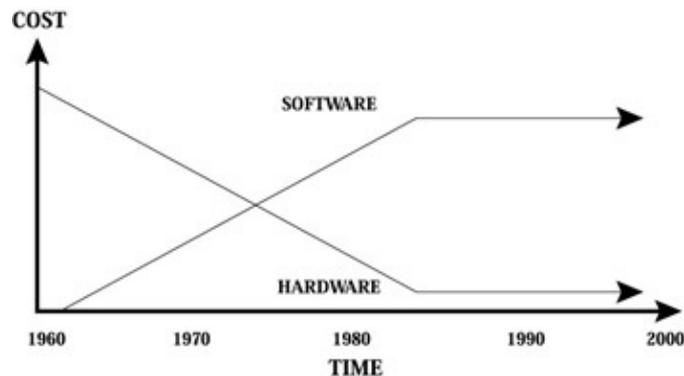


Figure 1–2: Cost of hardware and software trends with time

Since the inception of computers, there has been a desire for system development to reduce cost and delivery schedules while producing quality systems with few errors that contain system development goals and principles. Developers in the computer industry are continually striving to find an efficient method to achieve such goals. They have tried many methods and techniques, such as structured programming, structured analysis and design, and an object-oriented approach, but they have not achieved success.

Traditional methods, such structured analysis and design, contain functional decomposition, hierarchy charts, data flow diagrams, and state transition diagrams. Data-oriented analysis and design, events-oriented analysis and design, and their variations deal with operations and data as distinct and loosely coupled. Operations determine the structure of the system, and data are of secondary importance.

The object-oriented approach to system software development is preferable because it leads to construction of an application that consists of reusable classes and their objects. Objects are discrete software components that contain data and operation procedures together. Software systems are partitioned based on service objects, which interact between the client and server machines. Data determine the structure of the software.

Characteristics of a Good IT Project Manager

Personal Characteristics

- Is technically qualified
- Is a decision maker
- Is honest and creates a relaxed atmosphere
- Possesses the art of saying no without offending others
- Believes in managing time and people

Project–Related Characteristics

- Achieves the objectives and goals of the project within the established schedule, budget, and procedures
- Develops IT projects on budget and on time to the complete satisfaction of the users
- Has experience in related or similar projects
- Can control project outcomes by measuring and evaluating performance against established objectives and standards
- Develops and executes contingency plans to meet unforeseen circumstances and problems
- Develops and implements decisions relating to planning
- Is willing to redefine goals, responsibilities, and schedules as necessary to get the project back on track in case the schedule slips or the project is over budget
- Establishes and meets real priorities and deadlines
- Believes in good planning to reduce pressure and stress and increase productivity
- Establishes long–term and short–term planning

Team–Related Characteristics

- Has good communication and managerial skills
- Is able to plan, organize, lead, motivate, and delegate proper responsibilities to team members
- Respects team members and has their confidence and respect
- Shares success with the team members
- Selects the right person for the right job
- Shows appreciation to good workers
- Gets others in the organization to accept his or her ideas and carry out his or her plans
- Delegates duties and maintains control
- Believes in professionally training people for their delegated jobs
- Considers himself or herself as a part of the team
- Creates structured discipline
- Recognizes individual differences and takes advantage of individual strengths
- Provides work that stimulates a feeling of personal respect and professional growth
- Allows sufficient time for ideas to develop and mature
- Allows free time and encourages openness
- Understands the team members and creates effective communication
- Monitors his or her team members on a regular basis for the following types of people and takes necessary actions:
 - ◆ People who waste their time and that of others
 - ◆ Opportunists who steal others' ideas
 - ◆ Critical people who find only mistakes in others' work
 - ◆ Idle people who are unproductive
 - ◆ Egotistic people who brag about themselves
 - ◆ Gossips who spread rumors

IT Project Initiation

A project starts with the acceptance of a proposal by a customer who is willing to fund the project. A project can start within an IT organization with a statement of work or requirements by users. At this point the senior executives of the IT organization give a go–ahead signal to initiate the project. The executives select a suitable project manager who starts planning for the project.

Project-Related Characteristics

Planning covers coordination of available resources in an organization. The plan focuses on integration of system development goals and principles throughout an organization. To save cost and time, the plan includes adoption of a hardware and software reuse approach to systems development and maintenance throughout the life cycle. Changing old habits of a developing system via stovepipe practices is difficult. Practice of system reuse via the Internet requires time to change culture and mindset.

The manager develops a system infrastructure model, which is defined as the architecture of a domain. Domain architecture is defined as the high-level graphic model that concerns the main assets of a domain and their roles and relationships. The architecture specifies a generic description from the user's viewpoint and interface, input and output operations, memory organization, and control of a system. The architecture describes the nature, configuration, and interconnection of the major players of the domain. Architectures are necessary to develop reusable assets that adequately meet domain-specific requirements and are properly designed to interface and interact with other domain assets. Architectures are categorized as follows:

- Domain-specific architecture captures the commonality and variance of related systems within the domain and promotes the development of reusable assets to conform to those architectures. Domain-specific reuse exploits those domains to support building of the specific architectures.
- Process-driven architecture is the transparent assimilation of reuse and is an integral part of the system development process.
- Product line architecture is a set of systems that share a common infrastructure and satisfy the requirements of one or more domains. The product line approach involves the development and application of reusable assets for system development and maintenance among systems that possess similar infrastructures. Architecture centric is the reuse-oriented architecture in high-leverage domains to spur investment in the creation of reusable assets that comply with those architectures. The infrastructure of an organization manages reuse policies, standards, processes, business practices, roles and responsibilities, technology, training, and coordination across product lines.

Preparing for a cultural change is a continuous process in an organization. Almost everyone who has developed an IT system has practiced system reuse, and its benefits have been known for many years since the inception of computers. Much of that practice has been informal and ad hoc. Developers now recognize that the real payoff from reuse is achieved by moving to a more systematic approach. Preparations for cultural change are categorized as follows:

- **Paradigm changes** are defined as the overall structure of the process and are driven by the product line characteristics and the product line life cycle, organization readiness, and constraints.
- **Process changes** are defined as the activities and flows that fit into a given paradigm.
- **Method changes** are defined as the detailed techniques for activities within the processes.

Tools provide automated support to methods and processes. A reuse standard includes guidelines for requirements, reusability metrics, and certification.

Assessment of potential problems for implementation of system reuse in an organization is perpetual. Reuse is not a stand-alone technology. Developers are getting close to assessing problems that only raise more questions about reuse. The practice of reuse requires that reuse considerations be built into the system's development and maintenance paradigm together with the supporting processes, methods, and tools. Reuse is built into the system's development process, especially for a product line approach, because reuse decisions influence the architecture decision and requirement specifications. If developers introduce reuse in the later phases of system development, major reuse opportunities are likely to be precluded. Different types of domains may require different paradigms, processes, methods, and tools. Developers must define alternative practices and environments to accommodate different requirements.

Identification of Project External Interfaces

Standards and practices of system reuse establish the guidelines for use of reusable assets during system development and maintenance. These standards include the guidelines for practice during system requirements analysis, domain analysis, system design, software requirements analysis, software design, software implementation, documentation, repository management, and assets certification and validation processes. An organization establishes such a system reuse standard to be followed by all the developers. The requirement analysis standard establishes guidelines for reusable assets during the software requirements phase. During this requirement models phase, documentation is created in compliance with the established system reuse standard. The main features of requirement standards are the context diagram, requirements model, data dictionary, communication model, and domain identification.

Identification of Project External Interfaces

Identification of project external interfaces is important for design and estimation of resources and schedule. The context diagram (Figure 1–3) establishes external interfaces with other systems and objects. This diagram is a high-level view reflecting the project.

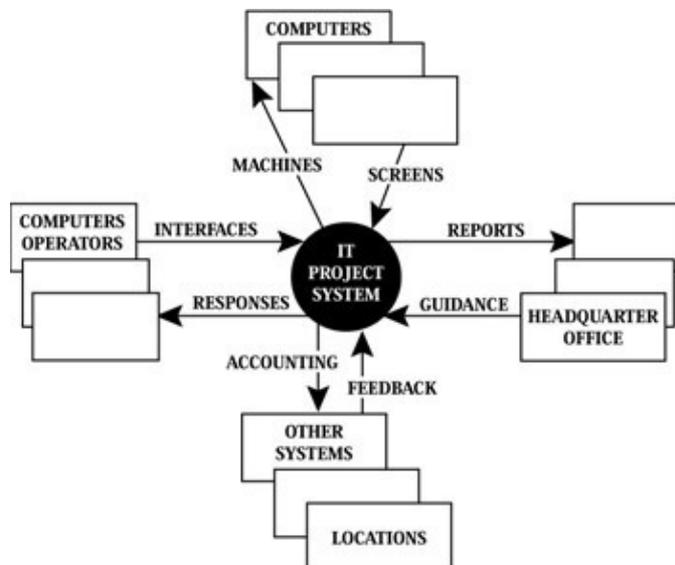


Figure 1–3: Context diagram

The system receives input to start from human operators. The system communicates externally with other systems and computers and submits reports to the headquarters office. To design a system, the developer must know with whom the system will be externally interfacing. The developer should state this information clearly in the system requirements to estimate resources and schedule.

Figure 1–3 shows that more than one human operator, computer, other system, and headquarters office can be involved. At this time the developer must know the external interfaces for the system under design.

Identification of Users, Customers, and Stakeholders

The IT project manager must identify the users, customers, and stakeholders at the beginning of the project. Figure 1–4 establishes the interfaces of users, customers, and stakeholders.

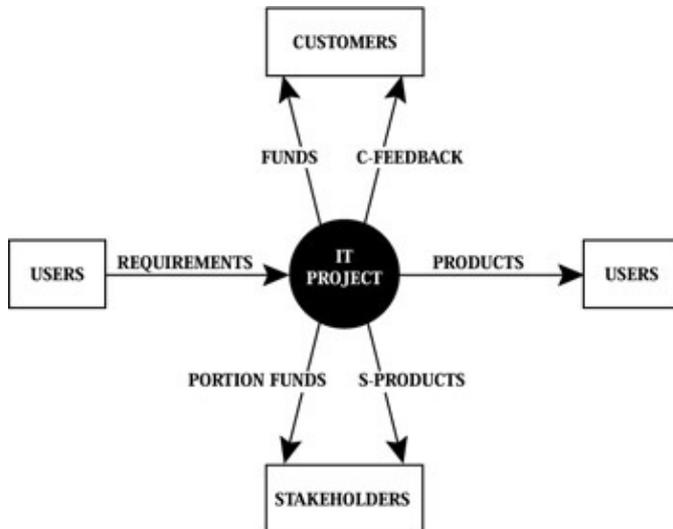


Figure 1–4: Relationship among users, customers, and stakeholders

Users

Users are those who provide requirements to design a system; they have need for that system and are going to use the system products. Users define their requirements and generate use cases to the system developers for testing of their requirements. Their involvement from the beginning of the project's development is necessary to reconfirm that their requirements are understood and that the system will be user friendly.

Users also state acceptance criteria. Acceptance testing involves the proper planning and execution of tests that demonstrate that the implemented system satisfies the requirements. The tests include functional, performance, regression, and stress tests. For their own satisfaction, users perform the acceptance tests to see for themselves that the result achieved is correct. Some of the tools used for the acceptance tests are the coverage analyzer, timing analyzer, and standards checker. Users detail the required documentation that is needed to support the system. The documentation relates to operation, training, and maintenance of the system. Users also require system design blueprints, hardware features, software design, source and object codes, and testing results.

Customers

Customers finance the project, which starts only if they have available funds. They have a vested interest in the project and desire a guarantee and warranty of the accuracy of the system. Customers also keep track of the system development to ensure that it is completed on time and within the schedule. Sometimes one person or organization can be both the customer and the user.

Customers and users form a team with IT system developers to monitor the progress. They ensure that all their requirements are accurately included in the system. The data dictionary documents the elements and definitions of interfaces, objects, classes, data groups, and data elements in detail. This document standardizes the process so that the system developer uses standardized terms and definitions throughout the system development process. This process enhances the quality and efficiency of the system.

Stakeholders

Stakeholders also have a vested interest in the IT project and provide their share of funds to complete the project. They will use all or part of the system products. They also generate requirements and use cases. Stakeholders form a team with users and customers to monitor the success of the project, participating from

IT Project Development Life Cycle Phases

the beginning to the end. They participate in all phases of system development and provide input for the success of the project. Stakeholders reuse the system products cost effectively.

IT Project Development Life Cycle Phases

IT project life cycle phases begin with the users' needs and requirements and continue until the successful completion of the project. The major life cycle phases include planning, prototype modeling, and the WBS of tasking, scheduling, milestone, budgeting, and system development. Most IT systems are driven by costly software; therefore the majority of time is allocated to software development and testing. Table 1–1 and Figure 1–5 illustrate well–managed IT project phases and allocation of resources.

Table 1–1: Allocation Of It Project Resources

Phase	Allocation (%)
Study and planning	8
System requirements analysis	6
System design	7
Infrastructure and prototype model	5
Reuse planning and Internet	6
Software requirements analysis	8
Software design	8
Software coding and testing	15
Software integration testing	10
System integration testing	15
System deployment	10
Audits	2

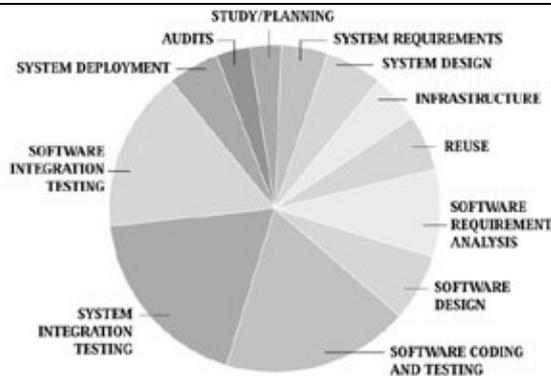


Figure 1–5: Major phases of an IT project

These percentage numbers are only for guidance. Depending on a project's size, the percentage of phases varies. Other variances are complexity of requirements, environment, unfamiliarity of software language and hardware, and manager and team member experience with the related projects. The manager provides the team with proper education and training that help in reducing risk and removing resistance among the team members.

Case Study 1–1

The following is the current method of ticket purchase at the Modern Airlines Corporation:

A passenger normally reserves the ticket and picks it up at the counter. At the ticket window, the passenger tells the ticket agent his or her name, either the flight number or departure time, and the destination of the flight.

The ticket agent enters the information into the computer terminal. When the system is 'up,' the computer finds the customer's reservation and generates a ticket. Once the ticket agent gets the completed ticket, he or she asks the customer for payment. The agent can only accept payment in cash, checks, or credit cards with proper identification.

The ticket agent notes the method of payment on a copy of the ticket that he or she files with the stack of purchased ticket copies. He or she writes the ticket number on the check, cash receipt, or charge receipt and files it in the payment drawer. The agent puts the ticket into a ticket packet and hands it to the customer.

The Modern Airlines Corporation wants to fully automate the ticket generation system for their customers at various airports. Bob is selected to manage this project because he possesses almost all the qualities of a manager that have been defined in this chapter. Bob follows the following steps to initiate the project:

1. He studies the project statement and tries to understand the project objectives, requirements, and goals.
2. He discusses the project with senior management and tries to understand the scope of the project, resources, schedule (24 months), and funding (\$400,000).
3. He identifies the customer, users, and stakeholders.
4. He draws a context diagram to show the roles of the customer, users, and stakeholders graphically (Figure 1–6).

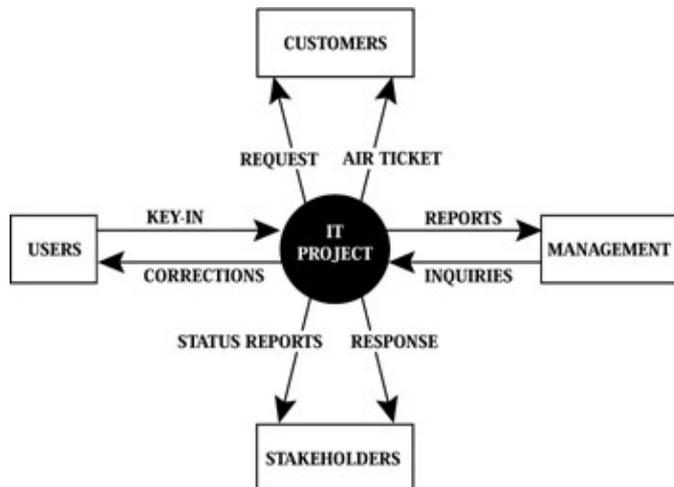


Figure 1–6: Generation air–ticket relationship among customers, users, stakeholders, and management

5. He sets up a small team among the management, customer, users, and stakeholders to discuss the progress of the project on a regular basis.
6. He allocates the money as follows:

Study and planning

\$32,000

Project Management Checklist

System requirements analysis	\$24,000
System design	\$28,000
Infrastructure and prototype model	\$20,000
Reuse planning and Internet	\$24,000
Software requirements analysis	\$32,000
Software design	\$32,000
Software coding and testing	\$60,000
Software integration testing	\$40,000
System integration testing	\$60,000
System deployment	\$40,000
Audits	\$8,000

This case study is revisited in later chapters.

Project Management Checklist

A good IT project manager must do the following:

- Identify users, stakeholders, and customers
- Communicate with users, stakeholders, and customers throughout the project
- Educate and train users, stakeholders, and customers about the project
- Understand the requirements of users, stakeholders, and customers
- Set up the project book
- Plan the project
- Staff the project
- Establish project phases
- Allocate the budget for various phases and activities
- Estimate resources
- Estimate the level of efforts
- Model and simulate the project
- Establish standards and controls
- Form project tasks teams
- Staff task teams
- Provide necessary training
- Staff support groups
- Provide a team concept
- Establish a schedule and milestones
- Provide a guideline to implement success
- Provide state-of-the-art tools
- Encourage the reuse of assets, the Internet, and telecommunications
- Track the deliverables
- Measure the success
- Create effective communication among the members of the project team
- Gain the users' confidence
- Show results
- Document the products
- Provide a smooth transition of the product to the users

Project Management Checklist

- Provide effective training that enables the users to use the product
- Evaluate achievements
- Pass on due credits to the staff
- Share profits and satisfaction
- Use the Internet to learn about related projects

Chapter 2: Guidelines for Successful Project Planning

Planning for an IT project covers coordination of management activities, phases, scheduling, timetables, and staffing. It includes the delegation of duties and responsibilities, exploration of modern technology, and system efforts in an organization. The plan focuses on integration of system hardware and software engineering and reengineering goals and principles throughout an organization. The plan also includes adoption of a reuse approach to a system's development and maintenance throughout the life cycle.

IT Project Planning

The IT project manager establishes a team that provides guidance in formalizing project planning. In Figure 1–5, only 8% of the total resources are allocated for the study and planning phase.

One of the guidance criteria is to establish system infrastructure and architecture and a reuse plan and standard at early stages. Most of the hardware components are industry standard and reusable. The manager must identify the domain and evaluate and procure a limited amount of hardware, software, and computer-aided software engineering (CASE) tools necessary to support the software reuse activities. The manager must establish a system reuse dialog among the practitioners and continue to educate them in system reuse that changes the mindset from stovepipe development to domain development. The project manager allocates separate resources for these phases.

Planning for an IT project covers coordination of management activities, phases, scheduling, timetables, and staffing. It includes the delegation of duties and responsibilities, exploration of modern technology, and system efforts in an organization. The plan focuses on integration of system hardware and software engineering and reengineering goals and principles throughout an organization. The plan also includes adoption of a reuse approach to a system's development and maintenance throughout the life cycle.

Project Book

The IT project manager generates and maintains the project book. The project book consists of clear definitions of the project and the steps taken to put all of the activities into action. The book contains all of the references used to inform decisions made in handling problems and contains limitations, meetings, reviews, checkpoints, and the overall outline of the progress of the project.

The purpose of the project book is to assist the project manager. The project book contains the following:

- A chronological record of events with dates
- A record of tasks and activities
- Organizational charts showing who controls and provides funding
- All correspondence in a chronological order
- Names of those with authority, executives, users, customers, and stakeholders controlling the project
- Plans and schedules for the project and its phases
- Standards by which to monitor the performance in each phase
- Reports to be reviewed
- Assessments of risk factors
- All recommendations for submission

Establishing Project Objectives

- A record of the implementation of necessary actions and changes that were undertaken for the project's success

The project book helps the manager in staffing each phase, scheduling, and projecting estimates. It is an ongoing record maintained during the course of the project, and the manager uses it as a critical document for evaluation of the project.

Establishing Project Objectives

As part of planning, the manager must establish the project objectives, which should include scope, constraints, and assumptions. The scope consists of boundaries and contours that shape the project. Constraints are the limitations of the project environment. Assumptions may be based on guarantees of assistance in completing the project.

The objectives included within management planning should be strategic, tactical, and operational. The strategy planning reflects the project goals and coincides with the corporate objectives. The project manager understands that the project goals are not achieved without determining, measuring, and completing the objectives.

The project manager properly records the objectives in the project book and openly discusses them with the concerned people. The objectives are clear, understandable, and followed by the members of the project team and should have the following characteristics:

- Objectives should be realistic.
- Management should make a firm commitment to them.
- Management should set up completion dates and costs to measure the achievement of committed objectives.
- Definite guidelines should show how to achieve the objectives.
- Manager should use effective communication to convey them.
- Objectives must be consistent with the available resources.
- Objectives must be clear, concise, and easy to understand and follow.

Setting Project Goals

The project manager sets the goal to make sure that the project is cost-effective; efficient; profitable; and completed within the time limit to the complete satisfaction of the users, customers, and stakeholders. The manager plans these goals on both short- and long-term bases. The project goals should include the following:

- Managerial effectiveness
- Efficiency
- Profits
- Quality products
- Effective communication
- Better services
- Maintenance of morale and positive attitudes
- Development of motivation
- Delegation of duties and responsibilities
- Establishment of controls, standards for raises and promotions, and status reporting
- Phases and completion schedules

Reporting Schema

- Project progress monitoring
- Realistic expectations

Reporting Schema

The project manager plans to set a schedule and a timetable for reporting. The manager identifies the personnel who are responsible for reporting to users, customers, stakeholders, and executives. The executives can be senior members of the organization, the Chief Information Officer, and financial executives. The manager also establishes interfaces with the Internet group and support teams. This kind of organization creates effective communication among the members of the team. This plan makes it easier for the manager to write status reports that are meaningful to the management and measure the project's progress efficiently.

The reporting schema plan assists the manager in determining who effects the change of scope during the phase and where to receive assistance when problems occur in the phase. The plan coordinates the measurement of actual versus planned performance, completion of the phase task, and expenditure of the budget. This plan also determines a proper balance of project team assignments.

Planning Project Recovery

The project manager establishes various types of recovery procedures and controls to manage the recovery of the project in case the project strays from the schedule or budget. This plan consists of implementing activities from the start of the project until it is completed. The manager selects and establishes standards, methodology, techniques, and tools to develop the system. His or her goals are to produce quality products on time and within the allotted budget to the customer's satisfaction. The manager achieves these goals by doing the following:

- Planning
- Setting specific goals and objectives
- Developing milestones
- Developing a prototype of the system
- Setting up an efficient organization
- Scheduling
- Monitoring the progress of the project
- Acting immediately to remedy variance from the set goals
- Providing status reports and feedback

The project manager sets up independent sections, such as quality assurance (QA), configuration management (CM), and testers, to assist him or her in achieving the overall goal. The establishment of these sections depends on the customer's requirements and the size and scope of the IT project.

Project Modeling and Simulation

Project modeling and simulation is a technique used to help the manager understand IT systems that are complicated, involve new technology, are embedded, and involve multidepartmental interfaces. *Model* is defined as physical, mathematic, or logical representation of a system. *Simulation* is defined as software implementation and solution of a model over time within the context of a scenario. The use of models, including emulators, prototypes, simulators, and stimulators, helps the project manager develop data as a basis for making a managerial or technical decision.

Reporting Schema

The project modeling and simulation is a convenient way to understand a system and develops a proposed solution to satisfy requirements. The system prototype model simulates some of the key characteristics of the final solution.

Figure 2–1 illustrates a typical system model structure. An extension of modeling is a project in which a first-cut solution is initially developed. The solution is further refined until it fully meets the customer's requirements. Prototype modeling and simulation is a convenient tool used to present a quick overview to the customer concerning the requirements and the result.

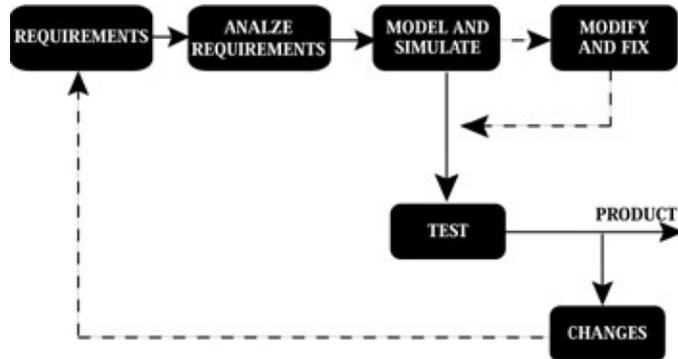


Figure 2–1: Modeling and simulation process

Modeling and simulation help the project manager understand systems that involve the following:

- Research
- Embedded programs
- Complicated requirements
- Multidepartmental interfaces
- New ventures
 - ◆ Scientific
 - ◆ Commercial
 - ◆ Business
 - ◆ Space applications
- Defense applications
- Government applications
- Academic research projects

The IT system's modeling and simulation process reduces time, resources, and risk. This process produces systems faster and cheaper and enhances quality. The modeling and simulation process also helps the manager do the following:

- Understand customers' requirements
- Analyze requirements graphically
- Manage tools
- Select hardware
- Select a system engineering methodology
- Select industry standards and best practices
- Outline the necessary documentation
- Describe test scenarios
- Allocate resources
- Define reusable assets
- Reduce the cost of development, testing, and maintenance

Work Breakdown Structure

- Increase efficiency
- Use reusable components effectively
- Simulate training

Work Breakdown Structure

The work breakdown structure (WBS) model depends on the customer's requirements as analyzed by the project manager. The system model provides a clear picture that enables the manager to understand the customer's requirements and define the WBS. This model includes the major events, activities, and tasks necessary for dealing with the IT system engineering product throughout the life cycle. The manager must plan the budget, schedule, estimate, and organization to achieve success for the WBS. Figure 2–2 illustrates a WBS model.

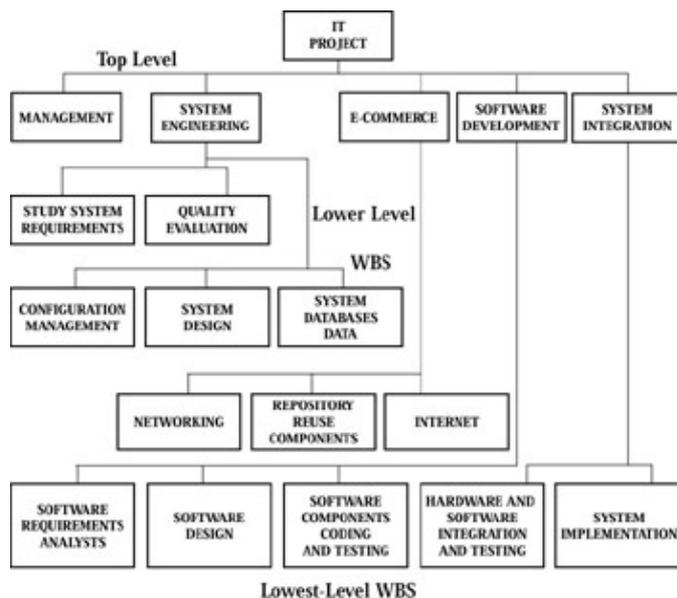


Figure 2–2: Sample WBS model

The system development center assists professionals by sharing insights learned by developing other system development products. This center maintains Internet access for reusable assets. In addition, an organization maintains a repository that can be accessed by other professionals.

Scheduling the Work Breakdown Structure

The project manager plans the schedule model for each WBS. This process establishes a schedule for the completion of activities and tasks. The project manager reviews and updates the schedule periodically. Some of the activities and tasks are completed in parallel, and others are achieved serially.

Scheduling helps the project manager maintain control over the progress of the system's development. The manager can update or modify schedules depending on the activity being delayed or successfully completed in a phase. The manager must take proper action in response to delayed activities to avoid such occurrences in the future.

The project manager develops a master schedule model to provide a comprehensive graphic picture of the priorities of various functions that must be completed. Figure 2–3 illustrates a sample master schedule. This figure shows WBS activities, tasks, and duration.

Milestone Model

IT Project	Assign To	2000 IFMAMJJASOND	2001 IFMAMJJASOND	2002 IFMAMJJASOND
Project A	Jack G.	Requirements Analysis System design	Modeling Reuse Software requirements	Coding testing Integration testing
Project B	Linda S.	Analysis and Design	Requirements Analysis and design	Modeling Reuse System design
Project C	Dan K.	Requirements Analysis System design	Modeling Reuse Software requirements Analysis and design	Coding testing Integration testing
Project M	Alice	Coding testing	Integration testing	

Figure 2–3: Sample master schedule

Model scheduling assists the manager in comparing the work achieved against the expected time frame, which creates better management control. The manager monitors deviation from the proposed schedule and corrects it more quickly. This helps the manager complete the system's development on schedule and within budget.

Milestone Model

The project manager develops a milestone model to accomplish a schedule. The milestone chart develops a graphic representation of a WBS effort organized according to time. The project manager documents this chart in the project book along with the analysis of the WBS, schedule, tracking of completion of activities, and tasks versus plan.

The earned value (EV) technique uses a milestone model:

$$EV = [ATD/EAC] \times 100$$

where ATD = actual to date and EAC = estimate at completion.

EV represents the percentage of completion of a project in terms of effort accomplished as a portion of total effort required. EV should not be identified in excess of 85% until the milestone is completed (i.e., 100%). This protects the project manager from the '99% complete syndrome' (the percentage of completion increases but never moves from 99% to 100% despite unlimited resources). For large, real-time IT systems, it is better to graphically represent cumulative EV in relationship to planned EV. EV is discussed further in Chapter 3.

Budgeting

A budgeting model estimates the cost of accomplishing a WBS as scheduled. The assigned budget allows expenditures so that the project manager can perform the different tasks, activities, system training, and phases effectively (Figure 2–4). The project manager determines the cost and schedule status and compares them with what is required to remain within the budget and schedule.

Project Organization

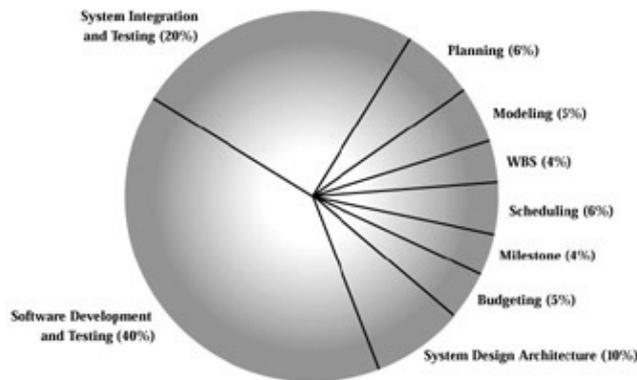


Figure 2-4: Budget model

The budget allows for expenditures such as the following:

- Hardware costs
- Networking
- Internet access
- Travel
- Material procurements
- Temporary living expenses
- Reuse assets
- Databases
- Contingency plans
- Resource procurements
- Commercial items
- Software development
- Support groups
- System testing
- Administrative expenses
- Personnel salaries and benefits

Many models are available to help the manager compute the cost estimation. Some of these are discussed in later chapters.

Normally a project manager agrees to a budget when he or she accepts duties and responsibilities for the phase. His or her duty is to manage the phase successfully within the time, schedule, and budget. The manager's expenditures should be consistent with the overall project budget and the assigned phase budget.

Project Organization

A manager plans the project organization model so that it conforms to the WBS, schedule, level of efforts (LOE), and budget. The manager hires human resources to meet the requirements and objectives at various phases during the system's development and maintenance.

The structure of an organization establishes relationships among members of the professional team, managers, stakeholders, users, and customer. The organizational structure distributes specific duties and responsibilities among the team members so they can function effectively.

The manager selects the right structure for his or her organization depending on his or her experiences. The most popular types of organization are the following:

Functional

- Functional
- Pure
- Matrix
- Hybrid

Functional

The functional organizational model defines hierarchy, technical competence, responsibilities, and duties (Figure 2–5). Personnel remain strictly within their assigned positions in the hierarchy. The project manager is ultimately responsible for decision making. The project manager assigns middle- and lower-level managers and supervisors to departments and sections, which are further subdivided into functional units headed by team leaders.

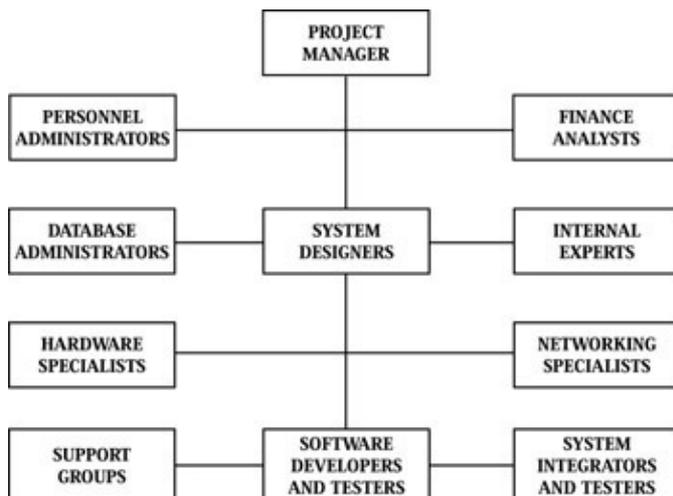


Figure 2–5: Functional organizational model

The advantages of functional organization include the following:

- Specialists in a field are grouped together.
- Control is centralized.
- Computer specialists are easier to manage.
- Training is improved.
- Morale improves.
- Mutual support is fostered.
- Professionals gain a variety of experiences from one another.
- Professionals are motivated to work.
- Policies and procedures are standardized and uniform.
- Productivity increases.

The disadvantages of functional organization include the following:

- Lack of coordination
- Competition
- Lack of individual responsibility

Pure

The pure organizational model encourages a straightforward management style (Figure 2–6). The project manager assigns all practitioners to a system project as one organizational unit. This structure model is also called a direct or vertical organization. The project manager is given authority over the system project and acquires resources from within and outside the organization as needed. The manager assigns practitioners to only one system project until it is complete. Although the system project organization promotes teamwork and a sense of unity, it can also lead to duplication of effort, problems with 'empire building,' and a tendency for project management to retain practitioners longer than necessary. In addition, competent practitioners move on when the system project is complete.

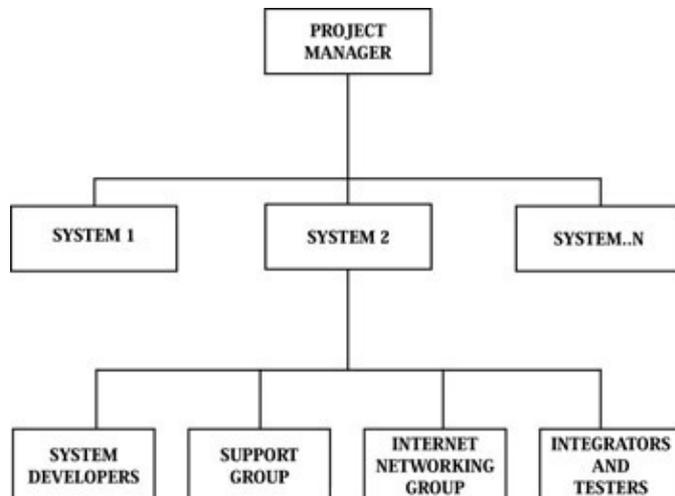


Figure 2–6: Pure organizational model

The advantages of pure organization include the following:

- It is easy to manage.
- Responsibilities lie upon one individual.
- It provides good control over resources.
- It encourages mutual communication.
- It is flexible to a project's orientation.

The disadvantages of pure organization include the following:

- Duplication of efforts
- Temptation to retain practitioners unnecessarily long
- Difficulties in keeping practitioners when the project is expired

Matrix

The matrix organizational model combines the best of functional and pure organizational styles (Figure 2–7). The matrix model is a multidimensional structure that tries to maximize the strengths and minimize the weaknesses of the pure and functional models. It balances objectives and provides coordination across functional departments and sections. The matrix model combines the popular hierarchic structure with a superimposed lateral or horizontal structure involving a project manager.

Hybrid

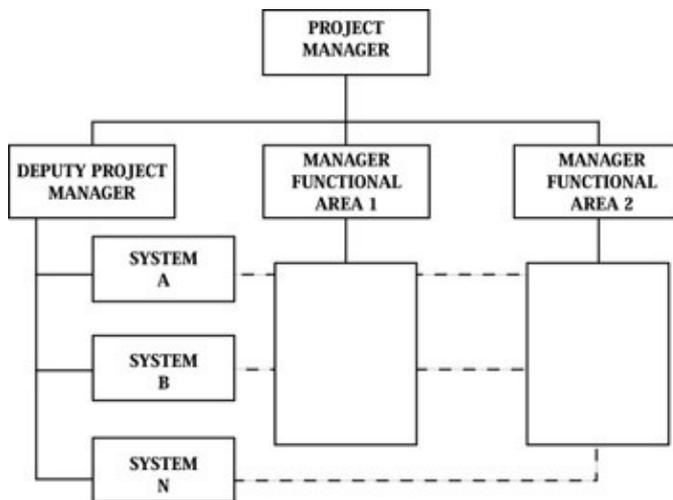


Figure 2–7: Matrix organizational model

The advantages of matrix organization include the following:

- Responsibility lies upon one practitioner.
- Professional experts are available from each of the functional departments.
- Project manager need not deal directly with personnel problems.
- Lower and middle managers solve project problems easily and quickly.
- Project personnel have job security.
- Personnel have job placement when a project is over.

The disadvantages of matrix organization include the following:

- Conflict of bosses
- Lack of management uniformity
- Limited authority for the project manager
- Low personnel morale
- Suffering productivity
- High turnover rate

Hybrid

The hybrid organizational model is a combination of everything that suits a manager's style (Figure 2–8). This organizational model does not have a formal format. The manager believes in the saying 'nothing succeeds like success.' No single approach is perfect for all IT project situations, especially for a complex system's development and maintenance. The best practices solution is contingent upon the key factors in the environment surrounding the IT project system's development and maintenance. The following are some of these key factors:

- New IT projects
- Conversion from the existing IT projects
- Maintenance of IT systems
- Prototyping of IT systems
- Complex IT projects
- IT projects for research and development

Staffing

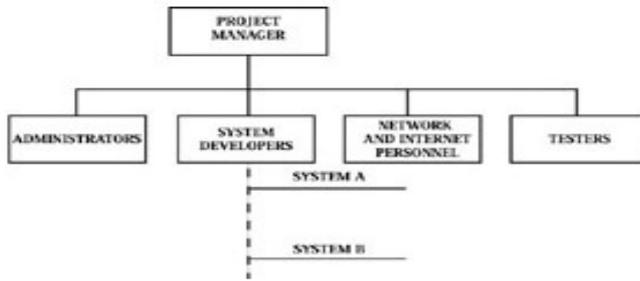


Figure 2–8: Hybrid organizational model

The advantages of hybrid organization include the following:

- It provides a hierarchic, structured approach.
- A maximum of five to seven practitioners supervise.
- It allows open–door communication.
- It allows management interfaces.
- It fosters encouragement of professional achievements.
- It allows individual responsibility for a function.
- It fosters encouragement of the team concept.
- It allows flexibility to suit the size of a project.
- It allows orientation toward domain–specific professionals.
- It allows an increase in productivity.
- It allows high morale.

The disadvantages of hybrid organization include the following:

- Lack of discipline
- Formation of cliques

Table 2–1 suggests criteria for selecting an organizational model.

Table 2–1: Criteria For Selecting An Organization Model

System Criteria	Functional	Pure	Matrix	Hybrid
Uncertainty	Low	High	High	Low
Importance	Low	Large	Medium	Medium
Technology	Standard	Complicated	New	New
Duration	Short	Long	Medium	Medium
Size	Small	Large	Medium	Medium
Time	Low	High	Medium	Medium
Complexity	Low	High	Medium	High

Staffing

Staffing is a management plan to establish a team suitable to the size of the IT project. Staffing and selecting personnel is a process that requires reviews of the IT system model, hardware and software development, and maintenance phases. A project manager is selected at the initiation of the IT project. He or she selects other

Staffing

personnel in the team gradually, depending on the WBS, schedule milestones, and budget.

The staffing model, as illustrated in Figure 2–9, consists of a plan to hire a set number of professionals in phases depending on the skills, duration, and degree of involvement in the project. The project manager assigns responsibilities and duties after careful consideration of each person's qualifications, experience, and motivation. The manager plans to provide the necessary training to suitable personnel to enable them to be efficient in their duties and responsibilities. The manager usually has difficulty finding the right professional for the right job within the budget. In the current competitive market, to find such professionals and retain them is a great challenge. The IT manager should prefer personnel who are loyal and have stayed with the organization for a long time.

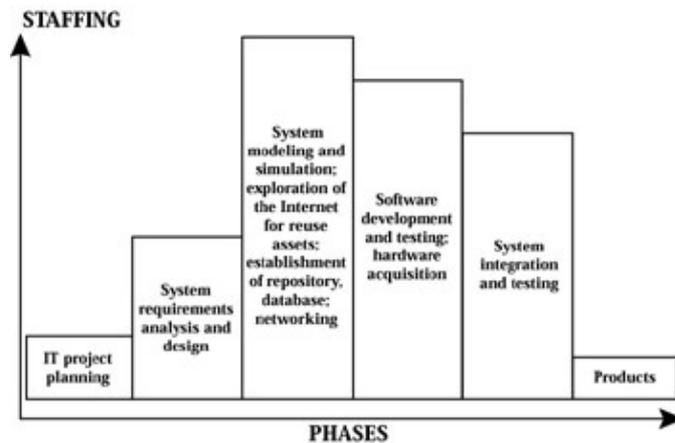


Figure 2–9: Staffing concept for a typical IT project

Shannon Brandon of MATRIX, an Atlanta-based firm that specializes in IT staffing, describes today's job market as 'frenzied.' With many more open IT positions than qualified candidates to fill them, the industry is rife with delayed projects, rising salaries, and frustrated IT managers. David Stum, president of Aon's Loyalty Institute, which researches levels of employee commitment and customer loyalty, points out that 'retaining employees is now a bottom-line proposition for American business.' The cost of finding, hiring, and training a new employee equals 6 months of annual salary and continues to increase (Wakin, 1999).

An education and training center helps the manager provide specific system and domain-related training. The manager should be a part of the team that provides training for his or her team members. The manager also should hire novice qualified people and have the center provide suitable training. Training of the professional is an ongoing effort; as the project progresses, suitable professionals should be trained to fulfill the requirements. This guarantees the creation of a team of professionals at less expense and with proper skills and disciplines for each phase of the system's development and maintenance. These trained personnel are knowledgeable and may stay longer with the organization, which helps an organization grow and helps the personnel grow in terms of raises and promotions.

The project manager carefully assesses the strengths and weaknesses of the personnel, especially for the assignments of phase, section, system, and task manager. These personnel have decision-making responsibilities and should be selected carefully to fulfill suitable roles. This is especially true for large IT projects in which the project manager needs to have a few phase managers who assist the project manager in management.

Revisiting Case Study 1–1

Planning is the most fundamental function of the project manager and includes decisions concerning the following:

- What to do
- How to do it
- When to do it
- Who will do it

As the 'Generating Air Ticket' project manager, Bob knows that project planning involves the following:

- Setting objectives
- Breaking the work into tasks
- Establishing schedules and budgets
- Allocating resources
- Setting standards
- Selecting a course of action
- Controlling and monitoring the project's progress

Bob knows that measuring and evaluating the project's progress is the key to success. He also knows how to identify problems early while there is still time to take corrective action. He establishes the project book to record all of the activities for future reference. Bob establishes controls, such as quality evaluation, CM, support groups, and a system development center. He involves users, customers, and stakeholders early in the project by establishing an integrated product team (IPT). He plans to set up tracking schema and establishes milestones for all deliverables. He keeps track of estimates and actual performance. He understands risk factor management.

Bob also understands the cost and schedule constraints imposed externally upon the IT system's development. He defines schedules and identifies milestones to monitor costs and assess quality.

Bob selects the matrix organization as best suited for his project. He sets up a team of consultants to start the system development center. The objective is to establish a modern technology mindset and educate personnel for the system development team. The team can accomplish a successful project if the manager and project team understand the design of the system with modern technology (e.g., reuse [plug in and play], Internet, networking, CASE tools, and commercial items).

Bob appoints three senior consultants to spend 2 months collecting data for understanding the system requirement analysis and project planning. They study other existing related systems by accessing the Internet. They prototype and build the system model as shown in Figure 2–10. They establish the task breakdown and the baseline for the project analysis.

Revisiting Case Study 1-1

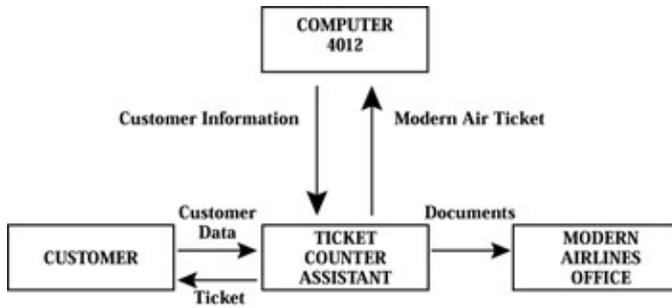


Figure 2-10: Generating ticket system's prototype model

Bob starts staffing the project in phases. He knows that it will take about 3 to 4 months to educate personnel in modern system development. It will take at least 1 year for a person to become an expert in the field. He lets his selected staff attend suitable regular classes that are conducted by the members of the system development center in the following areas:

- Microsoft Office software
- Internet
- Networking
- System engineering requirements analysis
- Software engineering design
- Object-oriented design methods
- Evolutionary design methods
- Modeling and simulation
- Testing
- Quality evaluation
- CM

Bob also establishes the precedent that the personnel need proper training before they can be considered productive and effective members of the project team. He initiates a monthly system development newsletter.

Bob develops WBS as shown in Figure 2-11. The WBS represents a sample of the task to be performed in a hierarchic structured form. The manager takes special care to not exclude any portion of the WBS that is of prime importance. The manager also avoids duplication of any portion of the WBS effort. The WBS also provides the work force and a budget allocated to different activities.

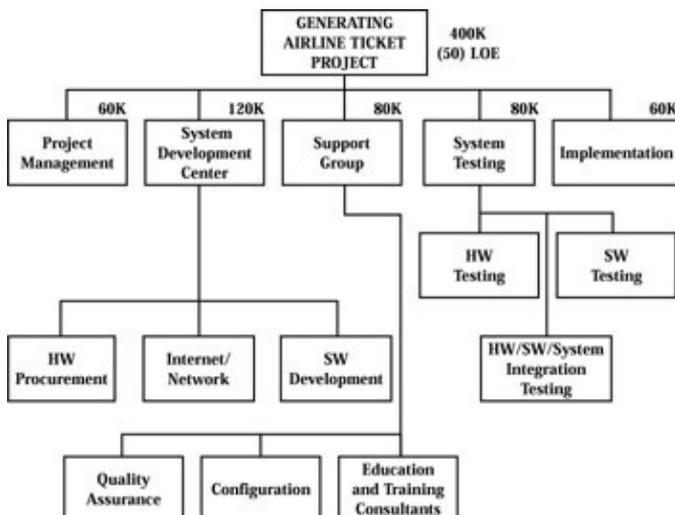


Figure 2-11: WBS, manpower, and budget allocation

Project Planning Checklist

Project planning is critical; the success of an IT project starts with structured planning. Success includes understanding requirements; selecting quality personnel; and using the correct methodology, hardware and software, and tools. Success also depends on achieving performance goals and reducing risk while implementing strategic objectives.

Technical professionals and managers must learn to communicate effectively. The project manager must try to establish a realistic schedule and goals. The manager builds in flexibility and contingency plans to save bogged-down projects. The manager sets up plans to gauge the project's progress, detect early warning signs, and determine how to handle them. The success of the project requires the manager to keep the IT project on schedule and budget. The project planning checklist includes the following:

- Set up a project book for future references.
- Establish project objectives, goals, and reporting schema.
- Establish a project recovery plan.
- Develop a project model to understand requirements.
- Get support from senior executives.
- Involve users, customers, and stakeholders early.
- Develop a WBS, schedule, and milestone models.
- Procure and allocate the budget properly.
- Select a suitable organizational chart.
- Hire appropriate staffing as the project progresses.
- Hire domain experts and share their knowledge and experiences.
- Use reliable technology that is matured.
- Establish a training center. Training is also a key to attracting and keeping IT professionals.
- Create a process to not only attract professionals but to keep them. The less appreciation an organization shows for a professional contribution, the greater the depreciation in staff loyalty.
- Establish an IT work force improvement program. What counts is not how long a professional has been with the organization, but how much he or she has learned and continues to learn.
- Remember that top performers know what they can do for an organization. They want to know what the organization can do for them.

Chapter 3: Project Estimating Techniques and Tools

Overview

The IT project manager must correctly estimate the following factors to achieve a successful, cost-effective IT project: cost, level of efforts, work breakdown structure (WBS), scheduling, staffing, milestones, budget, and IT system development and maintenance phases. Many techniques and tools assist the manager in such estimation. Each of the following tools is explained in detail in this chapter:

Cost-estimating tools:

- The constructive cost model (COCOMO) predicts the effort and duration of a project.
- The function points (FP) method measures the system development productivity.
- An earned value (EV) tool keeps a record of tasks and activities.

Scheduling techniques:

- The Gantt chart shows activities, their duration, and their interrelationships.
- The program evaluation and review technique (PERT) chart identifies the relationship of many steps involved in complex systems. PERT is a special application of network analysis for estimating and controlling the time required for activities and work packages about which little information is available from experience.
- A critical path involves activities through the critical path method (CPM) network. The CPM is a tool designed to reduce the length of time to complete a project.
- Microsoft Project 98 software is an automated tool that meets most of the IT manager's requirements in managing a project.

Cost-Estimating Tools

Constructive Cost Model

Barry Boehm developed the COCOMO in 1981 at the University of Southern California (USC). The COCOMO predicts the effort involved in and duration of a project. The COCOMO allows the project manager to estimate the cost, effort, and schedule when planning a new system software development activity according to practices that were commonly used in the 1970s and 1980s. It exists in three forms, each one offering detail and accuracy the further along one is in the project planning and design process. Listed by increasing fidelity, these forms are basic, intermediate, and detailed COCOMO. However, USC has only implemented the intermediate form in a calibrated software tool.

The implemented tool provides cost, effort, and schedule point estimates. It also allows a planner to easily perform 'what if' scenario exploration by quickly demonstrating the effect that adjustment of requirements, resources, and staffing may have on predicted costs and schedules (e.g., for risk management or job bidding purposes). The COCOMO needs the following parameters to compute the cost estimation of an IT project:

1. *Three classes of COCOMO projects*

- a. **Organic mode projects.** These are projects in which relatively small teams are working in a familiar environment developing well-understood applications. Communication overhead is

Chapter 3: Project Estimating Techniques and Tools

- low, team members know what they are doing, and they can quickly proceed with the job.
- b. **Semi-detached mode projects.** This mode represents an intermediate stage between organic mode projects and embedded mode projects. In semi-detached mode projects, the project team may be made up of experienced and inexperienced members. Team members may have limited experience of related systems and may be unfamiliar with some but not all aspects of the system being developed.
 - c. **Embedded mode projects.** Embedded mode projects are concerned with developing a system that is part of a strongly coupled complex of hardware, software, regulations, and operational procedures. Requirements are modified to get around software problems that are usually impractical, and validation and costs are high. Because of the diverse nature of embedded mode projects, project team members often have little experience in the application being developed.
2. **Projected length of the IT project software program.** This length is represented in thousands of lines of code (KLOC). For example, 75 represents a program with 75 KLOC.
 3. **Subcycles.** This number is the total number of subcycles. The maintenance subcycle is subcycle 1; therefore if the manager only wants the maintenance subcycle, he or she enters 1. Otherwise he or she enters the total number of subcycles, including maintenance. On the next level of input, the manager is prompted for more information about each subcycle.
 4. **Skill levels.** This number is the total number of different skill levels of personnel working on the project. On the next level of input, the manager is prompted for more information about each skill level.
 5. **Manpower.** For each skill level, the manager enters the number of people and the percentage of time that each skill level spends on each of the subcycles. For example, if five people from skill level 1 spend 50% of their time on subcycle 1, then the entries for subcycle 1 are 5 and 50.
 6. **Personnel information.** For each skill level, the manager enters the job title. For the salary field the manager enters the average annual salary for the skill level. The percent raise field represents the percentage that the skill level gets for a raise. As with other fields, it is in whole values (e.g., 12% = 12). The raise cycle is the number of months between each raise.
 7. **Subcycle information.** The manager enters the subcycle name and the percentage of time (in whole value) of the entire software development life cycle that is spent on this subcycle. All of the subcycle percentages must add up to 100%. The manager enters the overhead cost for this subcycle not associated with salary expense.
 8. **Maintenance information.** The subcycle name is automatically entered. The manager enters the duration in months and the overhead costs not related to salary expense.
 9. **Error message.** Valid values must be input for all fields. Title fields (e.g., job title, subcycle name) can be made up of any ASCII characters; all other fields must be numeric values.
 10. **Report output**
 - a. **Total cost.** The total cost reflects the overhead costs and the personnel (salary) costs for all of the subcycles, including the maintenance cycle.
 - b. **Total duration.** The total cost reflects the total amount of time (in months) needed to complete the project, including all of the subcycles and the maintenance cycle.
 - c. **Subcycle cost.** This value is the amount of money spent on employee salaries and overhead for a given subcycle.
 - d. **Subcycle duration.** The duration of a subcycle is the amount of time (in months) spent working on a given subcycle.

A simplified version of COCOMO is as follows:

$$MM = C (KDSI)^{\lambda}$$

Chapter 3: Project Estimating Techniques and Tools

MM = man-month, C = constant, K = thousand, DSI = delivered source instructions

The intermediate model of COCOMO consists of factors that affect the IT project cost and are called *cost drivers*. The following are some of these factors:

- Hardware compatibility
- The availability of computer-aided software engineering (CASE) and modern tools
- Staffing (domain experiences and training)
- IT project environment
- Management planning

The detailed model of COCOMO consists of the phases of software development, including all factors of the intermediate model. The phases depend on the standard and methodology to be used in the software development and maintenance life cycle. The major phases are stated as follows:

- System requirement analysis and design
- Software requirement analysis
- Software design
- Coding and unit testing
- Integration and testing
- System integration and testing

The manager estimates and applies the cost drivers to each phase separately. Ideally, the COCOMO estimated effort, MM(est), should match with the actual effort, MM(act), but this rarely happens. There is always a degree of error in computing practical applications. The percentage of error can be calculated using the following equation:

$$\text{Percentage of error} = [\text{MM}(\text{est}) - \text{MM}(\text{act})] / \text{MM}(\text{act})$$

The following are some of the effects of underestimating on a project (i.e., MM[est] is less than MM[act]):

- Understaffing
- Slipping schedules and milestones
- Rising costs
- Low morale

However, the following are some of the results of overestimating (i.e., MM[est] is greater than MM[act]):

- High costs
- Low productivity
- Professionals picking up 'slack habits'

The suggested solution is to compute a magnitude of relative error (MRE), which provides an absolute value:

$$\text{MRE} = \frac{\text{MM}(\text{est}) - \text{MM}(\text{act})}{\text{MM}(\text{act})}$$

A perfect score on the MRE test is an error percentage of zero.

COCOMO Example

A project is estimated to be equivalent in an effort to develop 33.2 KLOC of high-level language code. Let the project be of type organic (i.e., a simple application), and let all adjustment factors be nominal (i.e., the cost drivers or attributes are 'typical' of the environment for which the model was derived). Then use basic COCOMO as follows (COCOMO does not account for the concept phase costs and for early requirements, or about 6% to 10% of the front-end cost):

$$\text{Effort} = 2.4(\text{KLOC})^{**1.05} = 2.4(33.2)^{1.05} = 95 \text{ person-months}$$

$$\text{Duration} = 2.5(\text{Effort})^{**0.38} = 2.5(95)^{0.38} = 14.1 \text{ months}$$

$$\text{Full-time personnel} = \text{Effort/Duration} = 95 \text{ person-months}/14.1 \text{ months} = 7 \text{ persons}$$

The planner uses customized cost-drivers (weights) and financial and other constraints.

COCOMO II

COCOMO II is currently being developed to include modern approaches such as reuse, commercial items, object-oriented methods, and component-based system software development. COCOMO II is a model that allows the planner to estimate the cost, effort, and schedule when planning a new system software development activity. It consists of three submodels, each of which offers increased fidelity the further along one is in the project planning and design process. Listed in increasing fidelity, these submodels are the application composition, early design, and postarchitecture models. Until recently, only postarchitecture, the most detailed submodel, had been implemented in a calibrated software tool.

The new model (composed of all three submodels) was initially given the name COCOMO 2.0. However, after some confusion in how to designate subsequent releases of the software implementation of the new model, the name was permanently changed to COCOMO II. To further avoid confusion, the original COCOMO model was also redesignated COCOMO 81. All references to COCOMO found in books and literature published before 1995 refer to what is now called COCOMO 81. Most references to COCOMO published from 1995 onward refer to what is now called COCOMO II.

If the planner, while examining a reference, is still unsure as to which model is being discussed, a few clues are apparent:

- If the terms—basic, intermediate, or detailed (for model names) and organic, semidetached, or embedded (for development models)—are used in the context of discussing COCOMO, then the model being discussed is COCOMO 81.
- If the model names mentioned are application composition, early design, or postarchitecture, or if there is mention of scale factors, precedentedness, development flexibility, architecture/risk resolution, team cohesion, or process maturity, then the model being discussed is COCOMO II.

The major features of COCOMO II are as follows:

- Model structure, which consists of three models, assumes progress through a spiral-type development to solidify requirements, to solidify the architecture, and to reduce risk.

Exponent = Variable established based on rating of the following five scale factors:

PREC = Precedentedness

Function Points Method

FLEX = Development flexibility

RESL = Architecture or risk resolution

TEAM = Team cohesion

PMAT = Process maturity

- Size of object points, FPs, or source lines of code (SLOC)
- Cost drivers (the following 17 drivers must be rated):

RELY = Reliability

DATA = Database size

CPLX = Complexity

RUSE = Required reusability

DOCU = Documentation

TIME = Execution time constant

STOR = Main storage constant

PVOL = Platform volatility

ACAP = Analyst capability

PCAP = Programmer capability

AEXP = Application experience

PEXP = Platform experience

LTEX = Language and tool experience

PCON = Personnel continuity

TOOL = Use of software tools

SITE = Multisite development

SCED = Required schedule

Function Points Method

Allan Albrecht developed the FP method to measure IT project system software development productivity. The FP method is an alternative to estimating the LOC. The following are advantages of FP estimation:

- Helps the planner estimate LOC early in the life cycle
- Helps the planner estimate level of effort (LOE) easily

COCOMO and the Function Points Method

The steps involved in counting FPs are as follows:

- Count the user functions.

External input types

External output types

User interaction

Logical internal file types

External interface file types

External inquiry types

- Adjust for processing complexity.

The effort required to provide a given level of functionality varies depending on the environment. Once the planner computes the FPs, he or she can use them to compare the proposed IT project with past IT projects in terms of size.

Albrecht identifies a list of 14 processing complexity characteristics that are rated on a scale of 0 to 5, ranging from no influence to strong influence. The next step is to sum all the processing complexity points assigned. The number (n) is then multiplied by 0.01 and added to 0.65 to obtain weight:

$$PCA = 0.65 + 0.01(n)$$

where PCA = processing complexity adjustment (varies between 0.65 and 1.35).

Complexity factors vary between 0 and 5. This factor is then used in the final equation:

$$FP = FC(PCA)$$

where FC = function counts (previously computed).

The result is that the FPs can vary +/-235% from the original FCs, which are modified by complexity of the IT project. The FPs can be used to estimate LOC depending on the average number of LOC per FP for a given language. The FPs are subjective (i.e., they depend on the estimator). They cannot be counted automatically.

COCOMO and the Function Points Method

Two misconceptions about COCOMO, SLOC, and FPs are as follows:

1. COCOMO does not support the use of FPs. FP versions of COCOMO have been available since 1987. COCOMO II supports the use of either FPs or SLOC. In both cases, this is done via 'backfiring' tables of SLOC per FP for source languages at different levels.
2. It is irresponsible to use SLOC as a general productivity metric, but it is not irresponsible to use FP as a general sizing parameter for estimation. This breaks down into two cases:
 - a. If an organization uses different language levels to develop software, it is irresponsible to use SLOC as a productivity metric because higher productivity and SLOC are present at higher language levels. However, it is also irresponsible to use FP as a general sizing metric for

Earned Value Tool

estimation because pure FP will give the same cost, schedule, or quality estimate for a program with the same functionality developed using different language levels, which is clearly wrong. To get responsible results in this case, FP-based estimation models need to use some form of backfiring to account for the difference in language level.

- b. If an organization always uses the same programming language (level), it is responsible to use pure FP as a sizing metric for estimation. However, it is also responsible to use SLOC as a productivity metric (sunset.usc.edu).

Earned Value Tool

EV is an objective measurement of how much work has been accomplished on an IT project. *EV, performance measurement, management by objectives, and cost schedule control systems* are synonymous terms. EV improves on the 'normally used' spending plan concept (budget versus actual incurred cost) by requiring the work-in-progress to be quantified. Using the EV process, the IT project manager can readily compare how much work has actually been completed against the amount of work planned to be accomplished. EV requires that the project managers plan, budget, and schedule the authorized work scope in a time-phased plan. The time-phased plan is the incremental 'planned value' culminating into a performance measurement baseline. As work is accomplished, it is 'earned' using the same selected budget term. EV compared with planned value provides a work accomplished against a plan. A variance to the plan is noted as a schedule or cost deviation.

An EV method helps the project manager keep a record of tasks and activities plotted by sequence in time against actual versus planned completion. EV is computed as follows:

$$EV = ATD/EAC$$

ATD = actual to date and EAC = estimate at completion.

This represents the percentage of completion of a project in terms of effort accomplished as a portion of the total effort required.

Normally the established accounting system provides accumulation of actual cost for the project. The actual cost is compared with the EV to the indicated overrun or underrun condition. Planned value, EV, and actual cost data provide an objective measurement of performance, enabling trend analysis and evaluation of cost estimates at completion within multiple levels of the project.

An IT project manager should apply to every project where the owners of the final product wish to ensure that the expended resources were used efficiently. On major projects the application of good project management tools will aid in the selection of the right course when managers need to make financial and time allocation decisions.

Earned Value Management

Earned value management (EVM) is a management technique that relates resource planning to schedules and technical performance requirements. All work is planned, budgeted, and scheduled in time-phased planned value increments, constituting a cost and schedule measurement baseline. EVM is a valuable tool for identifying performance trends and variances from the management plan.

An EVM system has two major objectives:

- To encourage contractors (IT system developers) to use effective internal cost and schedule management control systems

Earned Value Tool

- To permit the customer (IT managers) to be able to rely on timely data produced by those systems for determining product-oriented contract status

The baseline plan in Table 3–1 shows that 6 work units (A–F) would be completed at a cost of \$100 for the period covered by this report.

Table 3–1: Baseline Plan Work Units

	Planned value (\$)
A	10
B	15
C	10
D	25
E	20
F	20
Total	100

Schedule Variance

As work is performed, it is 'earned' on the same basis as it was planned, in dollars or other quantifiable units such as labor hours. Planned value compared with EV measures the dollar volume of work planned versus the equivalent dollar volume of work accomplished. Any difference is called a *schedule variance*. In contrast to what was planned, Table 3–2 shows that work unit D was not completed and work unit F was never started, or \$35 of the planned work was not accomplished. As a result, the schedule variance shows that 35% of the work planned for this period was not done.

Table 3–2: Schedule Variance Work Units

	Planned value (\$)	Earned value (\$)	Schedule variance
A	10	10	0
B	15	15	0
C	10	10	0
D	25	10	–15
E	20	20	0
F	20	–	20
Total		100	65– 35 = – 35%

Cost Variance

EV compared with the actual cost incurred (from contractor accounting systems) for the work performed provides an objective measure of planned and actual cost. Any difference is called a cost variance. A negative variance means that more money was spent for the work accomplished than was planned. Table 3–3 shows the calculation of cost variance. The work performed was planned to cost \$65 and actually cost \$91. The cost variance is 40%.

Table 3–3: Cost Variance Work Units

Earned Value Tool

	Earned value (\$)	Actual cost (\$)	Cost variance
A	10	9	0
B	15	22	0
C	10	8	0
D	10	30	-15
E	20	22	0
F	-	-	-20
Total	65	91	-26 = - 240%

Spend Comparison

The typical spend comparison approach, whereby contractors report actual expenditures against planned expenditures, is not related to the work that was accomplished. Table 3-4 shows a simple comparison of planned and actual spending, which is unrelated to work performed and therefore not a useful comparison. The fact that the total amount spent was \$9 less than planned for this period is not useful without the comparisons with work accomplished.

Table 3-4: Spend Comparison Approach Work Units

	Planned spend (\$)	Actual spend (\$)	Variance
A	10	9	1
B	15	22	- 7
C	10	8	2
D	24	30	- 5
E	20	22	-2
F	20	-	20
Total	100	91	9 = 9%

Use of Earned Value Data

The benefits to project management of the EVM approach come from the disciplined planning conducted and the availability of metrics, which show real variances from the plan to generate necessary corrective actions (evms.dcmdw.dla.mil/index.htm).

Other Cost Estimation Tools

Other cost estimation tools are available in the industry. Some of these, like SECOMO and REVIC, are specialized toward unique methodology and language. ESTIMATICS is a proprietary product of Howard Rubin, and SLIM is a proprietary product of Larry Putnam.

Scheduling Techniques

The best known tools for scheduling and timing projects are the Gantt chart, the PERT chart, and the critical path method (CPM).

Gantt Chart

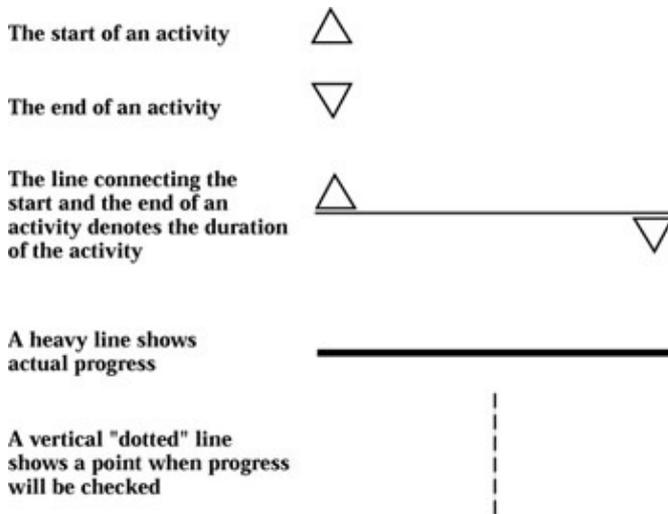
The Gantt chart, created around 1900 by Henry L. Gantt, is often used for scheduling. The Gantt chart shows activities, their duration, and the interrelationships among them. It is commonly known as the bar chart. The following are advantages of using the Gantt chart:

- It helps the manager plan well.
- It is easy to produce.
- The manager can compare work planned versus work accomplished at a glance.
- It is easy to understand.
- It provides a large amount of information on a single piece of paper.

The following are disadvantages of using the Gantt chart:

- It is difficult to depict interrelationships.
- It is weak in forecasting.
- It is inflexible.

The symbols used in the Gantt chart are shown in Figure 3–1, A, and a sample Gantt chart is shown in Figure 3–1, B.



3–1 A: Symbols used in the Gantt chart;

Activity	January	February	March	April
Activity 1	▲		▼	
Activity 2	▲			
Activity 3		▲		▼

3–1 B: Gantt chart showing project planning versus actual activity

PERT Chart

A PERT chart identifies the relationships among the many steps involved in complex projects. PERT was developed in 1959 in connection with the Polaris weapon system. The PERT chart is a special application of network analysis for estimating and controlling the time required for activities and work packages about which little information from experience is available.

A PERT network analysis consists of the following activities:

- Specifying the tasks to be completed
- Sequencing and interrelating work packages
- Setting up the network
- Scheduling

In PERT charts the events are connected by activities. Since events are end items (resulting products), they take no time, money, or resources themselves. However, activities require time, money, and resources. A sample PERT chart is shown in Figure 3–2.

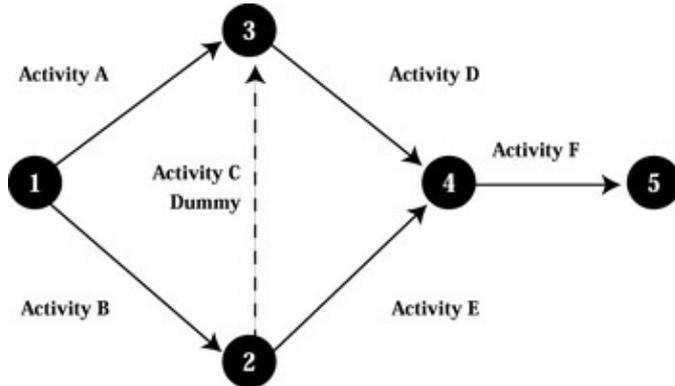


Figure 3–2: A typical PERT network showing activities that must be performed and their sequence. Note that the dummy activity has no duration.

The following are guidelines for drawing a PERT chart:

1. A person familiar with and committed to the project's objectives and requirements should develop the chart.
2. The developer starts with the ultimate targeted objective and proceeds backward to the beginning event.
3. The developer decides what activities must be completed before an event is completed.
4. An activity cannot begin until the event or end item preceding it has been completed.
5. The developer sets up two or more items and associated activities that can be accomplished concurrently in parallel paths.
6. The developer identifies a critical path to discover the longest cumulative time needed to accomplish end items and their associated activities.
7. The developer identifies a slack path to discover the shortest cumulative time needed to accomplish end items and their associated activities.

The following is a description of some of the conventions accepted by those who develop PERT charts:

- An activity is a time-consuming effort required to complete the project. It is represented by an arrow drawn from left to right. The arrows are not drawn to a time scale. Numbers above the arrows as shown in Figure 3–3, A indicate passage of time. By convention, an activity is designed by the node

PERT Chart

numbers at the beginning and end of the activity.

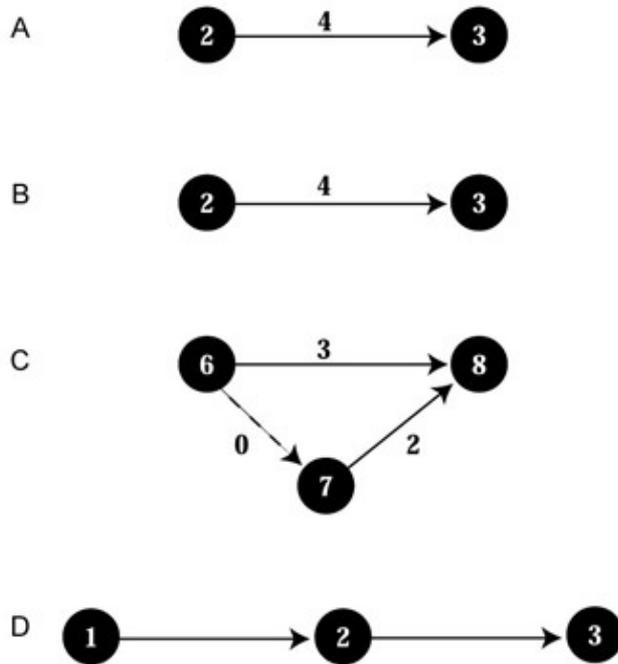


Figure 3-3: **A**, Activity; **B**, Event; **C**, Dummy; **D**, Restrictions

- An event is a particular instant in time that marks the beginning or end of an activity. Events do not require resources or time. A circle, called a *node*, containing a number, as shown in Figure 3-3, *B*, represents an event. An event is considered accomplished only after all the activities leading to it have been completed.
- A dummy is used to represent a restraint relationship that requires no time. It is indicated by a dotted line from the prerequisite activity to the beginning of the restricted activity, as shown in Figure 3-3, *C*. It is treated as an activity with zero time.
- A restriction is a prerequisite relationship that establishes the sequence of activities as illustrated in Figure 3-3, *D*. Activity 1-2 is prerequisite to activity 2-3. A prerequisite activity immediately precedes the activity being considered. Activity 2-3 is postrequisite to activity 1-2. A postrequisite activity immediately follows the activity being considered. Figure 3-4, *A*, shows that activities 2-5, 5-7, 2-4, 4-6, 3-7, and 6-7 are descendants of activity 1-2. A descendant is an activity restricted by the activity being considered. In Figure 3-4, *B*, activities 1-2 and 2-5 are antecedents of activity 5-7. An antecedent is an activity that must precede the activity being considered.

PERT Chart

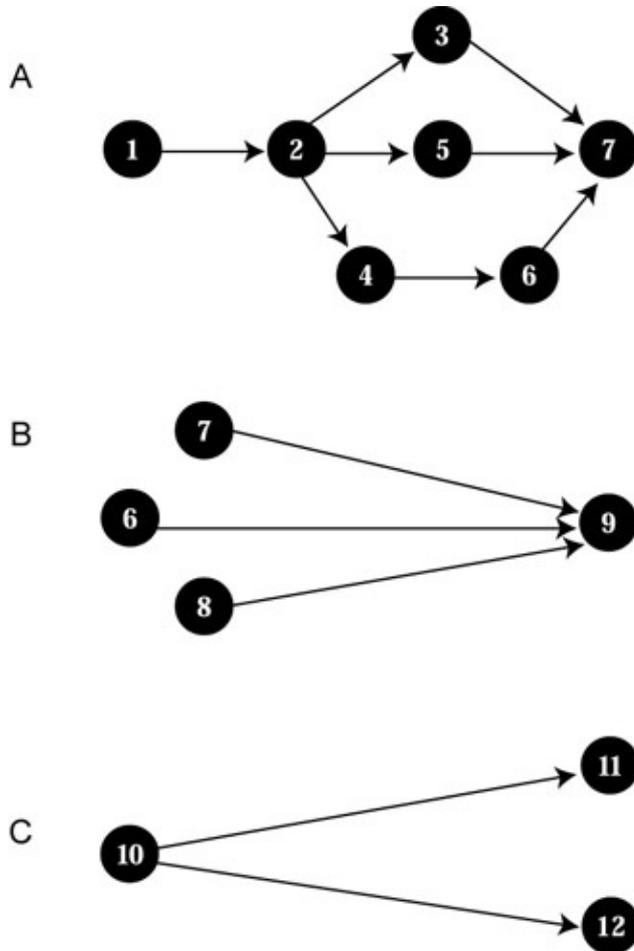


Figure 3-4: A, Another example of restrictions; B, Merge; C, Burst

- A merge exists when two or more activities converge at a single event. All activities merging at an event restrict all activities beginning at that event as shown in Figure 3-4, C. The event is not considered to occur until all the activities merging at the event have occurred.
- A burst exists when two or more activities begin at a single event as shown in Figure 3-4, C. All activities have a common prerequisite. Merges and bursts occur when concurrency exists.
- An arrow network is a graphic representation of the activities and events needed to complete a project. The network shows the logical relationships existing among the various activities as shown in Figure 3-5. Conventionally, networks are drawn from left to right. No two activities may have the same starting and ending events; use of dummies eliminates the problem. It is customary to number the events in such a way that the event at the right of the activity arrow has a higher number than the event at the left of the arrow. The lengths of the activity arrows have no meaning and may vary to provide clarity to the network. Crossovers are permitted but should be avoided wherever possible. Loops are not permitted (i.e., no path subsequent to an event may lead to a prior event).

PERT Chart

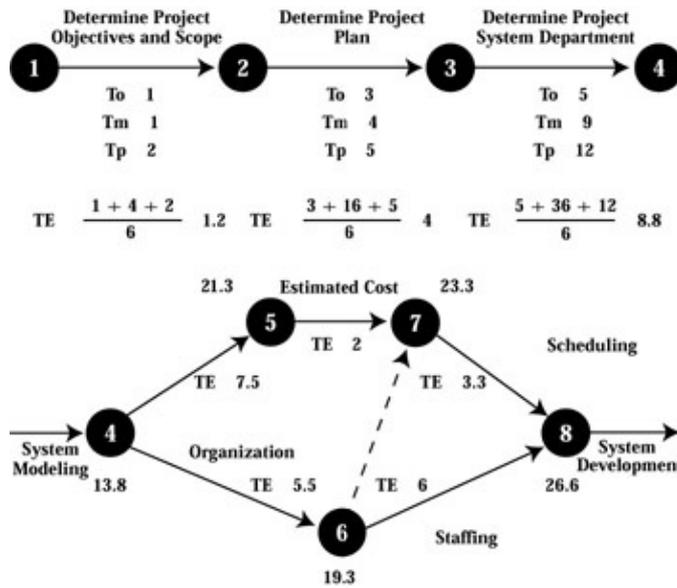


Figure 3-5: A PERT network

- For PERT activities, time expected equals three times the average estimate. Estimates are required for all the activities of the network to estimate the total time:

$$\text{Time expected (TE)} = \frac{T_o + 4T_m + T_p}{6}$$

T_o = optimistic time (an estimate of the minimum time an activity will take if unusually favorable conditions are experienced), T_m = most likely time (an estimate of the normal time an activity would take if it was repeated an indefinite number of times under identical conditions), and T_p = pessimistic time (an estimate of the maximum time an activity will take if unusually unfavorable conditions are experienced).

- The manager finally calculates an array of estimates regarding different durations for the project.

The following are advantages of using PERT charts:

- The use of the three time estimates makes it more valid than any other technique.
- It is easy to calculate the effect of delays.
- PERT charts are strong in the system development and maintenance phases of scheduling.
- The manager can see the activities that must be completed before end objectives can be met.
- They are good for simulating the influence of various resource allocations on the schedule.
- They are strong in forecasting whether the manager will accomplish future events on schedule.
- They predict slippage and their effects early enough for the manager to take remedial action.
- Activities are clearly identified, and elapsed times can be obtained as needed.
- It is easy to adjust time estimates when changes occur.
- It is easy to plan and schedule work.

The following are disadvantages of using PERT charts:

- They are complex.
- Estimation of activity times is time consuming.
- They are expensive.

Critical Path Method

The critical path is the longest path through the CPM networks. Morgan R. Walker of Dupont and James S. Kelly of Remington Rand developed the CPM in 1957. The CPM predicts the duration of the project. When a project is delayed, this is the critical path through which to go. All other paths are called *noncritical paths*, *slack paths*, or *float paths*. These paths are the difference in duration between two nodes.

Nodes are the places where two or more activities are starting or ending (see circle 3 in Figure 3–2). The activity is a distinct part of a project. In Figure 3–2 the project has six activities:

1. Activity A: 1 to 3
2. Activity B: 1 to 2
3. Activity C: 2 to 3
4. Activity D: 3 to 4
5. Activity E: 2 to 4
6. Activity F: 4 to 5

An event is a specific point in time and is shown by a circle. This is the start or completion of an activity. A dummy activity is an activity that has no duration. It is used only to show the precedence of activities and is shown as a dotted line. In Figure 3–2, activity C from 2 to 3 is a dummy activity.

The following are advantages of using CPM:

- It is good for single-shot activities.
- It is excellent for simulating alternatives plans.
- It forecasts strongly so that future events can be accomplished on schedule.
- It allows for clear identification of activities.
- It allows for easy estimation of time.
- It is easy to change the graphics to reflect program changes.

The following are disadvantages of using CPM:

- It does not provide a formula to estimate time to completion.
- It is easily prone to error.
- It is weak in the system software development phases.
- It is not good for scheduling.
- It is a complicated tool for planning and status reporting.

Case Study 3–1

The Modern Project Management Corporation (MPMC) decided to reengineer an existing IT system. They need new hardware, including laptop and desktop computers, networking, and Internet access. They also need the necessary software. You have been selected as manager to plan this project. The following are suggestions on how to begin:

1. Compile the data in tabular form as shown in Table 3–5.

Table 3–5: Case Study Date

Critical Path Method

Activity	Predecessor	Successor	Duration (days)
A. Plan project	None	B, C, D	5
B. Acquire hardware	A	E	10
C. Select location	A	E	6
D. Determine needed amenities			
<ul style="list-style-type: none"> ◆ Electricity ◆ Telephone ◆ Workstation ◆ Software ◆ CASE tools ◆ Office furniture 	A	F	7
E. Install hardware	B, C	F	4
F. Provide amenities	D, E	G	2
G. Start functional			
<ul style="list-style-type: none"> ◆ Early start (ES) ◆ Late start (LS) ◆ Early finish (EF) ◆ Late finish (LF) 	F	None	2

- Draw the CPM network and determine the early start (ES) and early finish (EF) for each activity, moving from the start to the end of the project, as shown in Figure 3–6. Place the number zero in the ES position of the first activity. The EF of the activity is equal to its ES plus the duration of the activity. Place in the ES position of each successor activity the EF of its predecessor. If an activity has two or more predecessors, use the largest EF. Continue until the EF of the last activity of the project has been calculated.

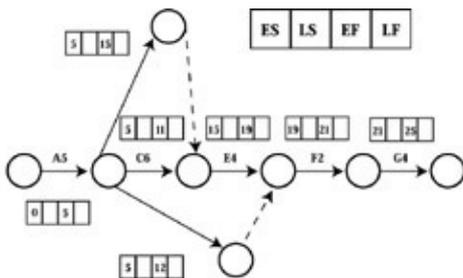


Figure 3–6: Project activities

- Determine the late start (LS) and late finish (LF) of each activity, moving from the end to the beginning of the project, as shown in Figure 3–6. Place in the LF position of the last activity the number that is in its EF position. The LS of the activity is equal to its LF minus the duration of the activity. Place in the LF position of each predecessor activity the LS number of its successor. If an activity has two or more successors, use the smallest LS. Continue this process until the LS of the first activity of the project is determined.
- Determine the LS and LF of each activity, moving from the end to the beginning of the project, as shown in Figure 3–7. Place in the LF position of the last activity the number that is in its EF position. The LS of the activity is equal to its LF minus the duration of the activity. Place in the LF position of each predecessor activity the LS number of its successor. If an activity has two or more successors, use the smallest LS. Continue this process until the LS of the first activity of the project is determined.

Critical Path Method

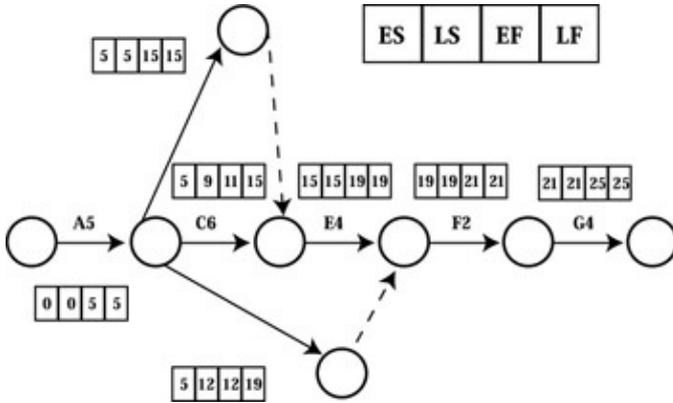


Figure 3-7: Project activities CPM

5. Determine the longest path through the CPM network, or the critical path: ABEFG = 25 days
6. Draw the Gantt chart for the project as shown in Figure 3-8.

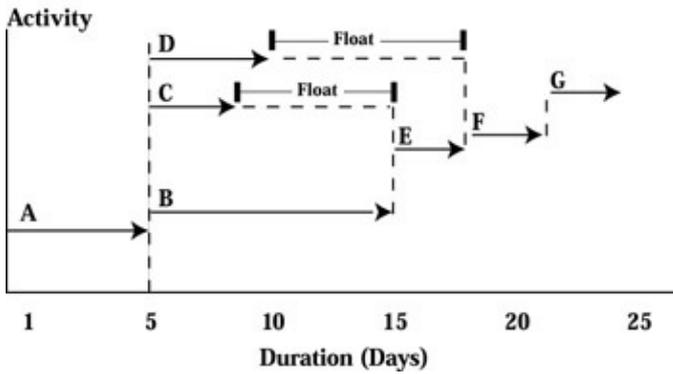
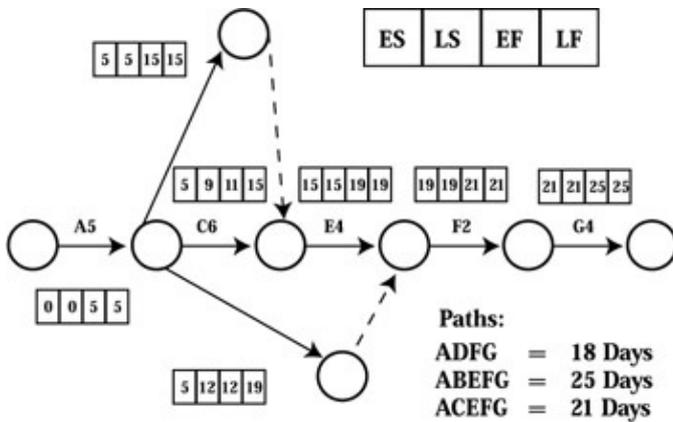


Figure 3-8: Project activities and Gantt chart

Chapter 4: Project Management Quality Control Mechanism

The IT project management quality control mechanism involves the establishment of efficient system development practices and monitors those practices throughout the life cycle. This results in the development of high-quality, cost-effective IT system products. The IT project manager should plan to provide and enforce the quality principles. The manager establishes the quality assurance section (QAS) to control the established quality principles. The manager makes it clear that the quality is the responsibility of all those who are involved in the system development and maintenance and not just the QAS.

System Quality Management

System quality management is the prime responsibility of the IT project manager. The manager establishes a quality assurance (QA) team during the initial stages of project management planning. The QA team forms a section and assigns duties and responsibilities to their members who will measure the quality parameters at various stages in the system development and maintenance. The QAS members control the quality according to the established standards and procedures. Their responsibilities include the following:

- Prepare the QA plan
- Attend system hardware and software meetings, walk-throughs, and reviews to ensure that the products do what they are supposed to do and conform to the established quality parameters
- Measure the products with the established standards and procedures
- Evaluate tests

Test plans

Test procedures

Test scenarios

Use cases for testing

- Evaluate deliverable documents
- Submit recommendations to improve the quality
- Report the findings to the project manager
- Establish and follow the QAS checklist and guidelines
- Evaluate quality

System Quality Characteristics

Quality is defined as the presence of desired characteristics and the absence of undesirable characteristics in the IT system's products. The manager decides about these characteristics when consulting with customers, users, and stakeholders.

The manager must identify and measure the quality characteristics of each system product during the IT system development phases. This starts during the system requirement analysis stage, continues through each successive phase, and culminates in the operation and maintenance phases. Quality deals with how well the system product performs its functions in the operating environment over time. Each characteristic needs metrics to measure quality. Metrics include indicators that provide an early warning for detecting the system's

quality problems.

Quality Metrics

Quality metrics provide a quantitative measure of the degree to which the system possesses and exhibits certain quality characteristics. QAS metrics are as follows:

- Management metrics, which include quality, cost, and schedule
- Process metrics, which include effectiveness and efficiency
- Product metrics, which include defects, complexity, and maintainability

The overall system quality is a function of the metrics values. The product functions are summarized as hardware and software failure, reliability, recovery, modifiability, availability in time, and maintainability. Suggested quality metrics is the degree to which the system is portable, accurate, complete, robust, consistent, accessible, and structured:

- IT system software is portable if it can be operated easily on a computer configuration other than its current one. Its portability consists of hardware independence, initiation of its own storage area, and use of data types that are hardware independent.
- A system is accurate if it meets the customer's demand for accuracy. The system does what it is supposed to do in accordance with the requirements. This confirms that the development of hardware and software followed established IT standards, data typing, timing constraints, and runtime parameters.
- System software is complete if all of its components and units are present in a computer software configuration item (CSCI) and hardware configuration item (HWCI). Each component is fully developed with the established standard and well documented. This requires that the system software be well-commented and include a graphic model, interfaces, and an interoperability network.
- System software is robust if it continues to perform despite a violation of assumptions in its specification. Some of the precautionary measures for robust system software are error-handling concepts, error exceptions, input data for range errors, and availability of default values.
- System software is consistent to ensure uniform notation, terminology, and symbology. It is consistent internally and externally to the extent that the content is traceable to the requirements, forward and backward. The manager achieves consistency by use of pretested generic packages, name-of-data dictionary elements, objects, database files, reusable modules and packages that are defined in the previous phases of the system software development, sets of global variables, a common area, and real and integer data types.
- System software is accessible to facilitate the selective use of its components for reusability, efficiency, and testability.
- System software is structured when it involves a definite pattern of organization of its components, follows a systematic approach, follows set standards, and is uniform and easy to understand and follow.

Quality Indicators

Quality indicators provide insight into the adequacy of the system software specifications. The manager can use quality indicators effectively throughout the system life cycle. They lay the foundation for quality software development, which is crucial to obtain a reliable and maintainable IT system. The manager can use the values determined from the components of the completeness indicator to identify specific problem areas within the system software specifications and design.

Quality Parameters

Quality indicators assist in identifying the following:

- How well system level requirements have been translated into software requirement specifications and hardware requirement specifications
- Any hidden or missing linkages in the system design
- How well the customers', users', and stakeholders' requirements have been translated into the design
- The degree of implementing testable software products
- System stress during testing
- The percentage of hardware and software errors detected during system testing
- System documentation meeting the need of the customer, user, and stakeholder in maintenance

Quality indicators can be summarized as follows:

- **Completeness.** Depends on the number of defined functions, data items, referenced functions, decision points, condition options, calling parameters that do not agree with the defined parameters, and the data references that have no destination
- **Design structure.** Depends on the number of system software packages dependent on the source of input or destination of output, the number of system software packages dependent on prior processing, and the number of database items
- **Detection of defects.** Depends on analysis of the system requirement, system design, hardware components, software requirement, software design, and code inspection process
- **Detection of faults.** Depends on test results and the analysis of the system requirement, system design, hardware components, software requirement, software design, and code inspection process
- **Test coverage.** Depends on the system structure and system requirements
- **Test sufficiency.** Depends on the number of faults, system components integrated, software packages in a CSCI and HWCI, and faults detected to date
- **Documentation.** Depends on system software and hardware documentation in accordance with established standards

Quality Parameters

The IT system manager establishes quality parameters to measure the quality at various phases and stages of the development and maintenance of the system. The manager states the methods and techniques used to maintain quality during the development of the system plan and in response to the request for proposal. The manager establishes the quality parameters to measure the quality of the following items:

- Hardware development phases
- Software development phases
- Established standards
- Guidelines and procedures
- Products developed at each phase
- Walk-through
- Reviews
- Inspection
- Monitoring of the progress
- Enforcement of standards and procedures
- Deliverable hardware, software source, and object code
- Deliverable document

Quality is a relative term. The quality changes with changes in the quality parameters. The following are the quality parameters that the system developers should check:

Technical Reviews

- Accuracy
- Correctness
- Reliability
- Testability
- Maintainability
- Portability
- Reusability
- Efficiency
- Modularity
- Ease of use
- Understandability
- Adherence to standards
- Error-free products
- Traceability

Technical Reviews

A technical review is a disciplined examination of a document that is extensive and provides critical evaluation. The technical review helps developers do the following for an IT project:

- Find errors in the system development in early stages
- Track the progress of the project
- Provide feedback
- Educate the reviewers
- Pass the products of each phase to the next phase

The manager can base technical reviews on either the established standards or a methodology defined by his or her organization. A checklist in the standard guides the preparation of review participants and helps the professionals organize and conduct reviews. The participants in a technical review consist of a review leader, a recorder, a group of reviewers, and producers of documents. The reviewed document is either accepted or rejected in a review. The experiences of the participants attending a review provide feedback concerning the correctness of the document. The QAS attends the review to ensure the quality and correctness of the document in accordance with the system's requirements.

The QAS also ensures that all action items on the review report are addressed. The QAS analyzes the data on the review forms and classifies defects to improve the system development and review processes. These data include error-detection efficiency, cost-effectiveness, reliability assurance techniques, and tools supporting the reviews.

Technical Walk-Through

The technical walk-through determines and ensures the correctness and quality of the documentation. The manager should conduct it at all phases of the system development and maintenance. Whether formal or informal, technical or nontechnical, a walk-through creates a friendly, professional atmosphere different than that of a meeting and review. The result of a walk-through is the creation of an error-free, cost-effective document that performs its required functions.

The QAS ensures that technical walk-throughs are conducted as many times as needed to create a quality product.

System Quality Improvement Process

The system quality improvement process continues during the life cycle of the IT system development and maintenance. The members of QAS are quality professionals with many years of experience in hardware and software development. They should be trained to know more about IT systems than other developers. The project manager should select for QAS those professionals who believe in quality and keep on learning and practicing quality. Sometimes it is better to rotate developers from the QAS to other sections of the system development to gain a variety of experiences. This also gives other professionals a chance to learn more about quality. The following are the characteristics of a good QAS professional who can improve the quality process:

- Believes in quality
- Improves QA discipline
- Shows experiences in system development and maintenance phases
- Has experience to introduce quality in documenting the IT system
- Has a positive and innovative attitude
- Supports the quality organization's QA program
- Interprets guidance and directives for the QAS
- Prepares QA policies and procedures
- Applies imagination and creativity for improving the system's quality
- Shows responsibility and dependability in improving quality
- Complies with quality policies and guidance
- Is sound in technical accuracy and completeness
- Has good interpersonal relations skills
- Is good at representing the organization at system reviews and meetings
- Has good communication skills
- Performs walk-throughs, reviews, inspections, and internal audits

The system quality process starts during the planning and understanding of the system requirements and cumulates at the completion of the project. The QAS members conduct research and study parameters to improve the quality of the system.

System Software Quality and Interoperability

The project manager forms a system software quality and interoperability working group. This working group considers how to best apply electronic commerce (e-commerce) off-the-shelf products to enhance their operations as part of their e-business initiative. The working group develops series of questions that provide the IT manager with information for better understanding of e-commerce and e-business software and interoperability. The working group covers the following questions for software quality, interoperability with hardware and software, and management of user expectations (www.E-Interop.com):

A. Software quality

1. What are the most important attributes that define quality in your software product?
2. What is your organization's system software development process in the following areas?
 - a. Requirement determination (How do you decide what functions will be included in the software product?)
 - b. Testing and validation program
 - c. QA program

System Quality Improvement Process

- d. Creation and validation of technical documentation
- e. Relation of product function to user training
3. How does your organization Ensure [InsideOut] software quality?
4. Do you measure quality by Using any of the following indicators or benchmarks?
 - a. Certification by independent laboratories
 - b. Number of calls for technical support
 - c. Number of errors found by customers
5. How does your organization address risk management for a software product?
 - a. How does your organization identify potential risk factors and assess their possible influences?
 - b. What is the tolerance for potential risk factors? Are there general guidelines for the amount or level of risk or liability that your organization is willing to accept?
 - c. c. What are your risk mitigation and/or risk control activities?
6. How does your organization measure customer satisfaction for your software product, documentation, support services, and user training?
7. When you organization makes purchasing decisions for system software, what do you look for? Rank the following criteria in order of importance:
 - a. Price
 - b. Robust functionality
 - c. Reputation for quality
 - d. Interoperability

B. Interoperability with hardware and software

1. What are your organization's interoperability needs to support e–business? How are they expected to change in the future?
 - a. Characterize the communities with whom interoperability is required (e.g., internally, externally).
 - b. Characterize the nature of the interoperability required with each community. If appropriate, consider interoperability requirements from two perspectives: minimal level (must have) and desired level (would like to have). How important is it to get beyond the minimal level?
2. To what extent and how has your organization solved interoperability (internally and externally) requirements to enable e–business with respect to the following?
 - a. Data and document interoperability (discovery, access, and semantics)
 - b. Application interoperability
 - c. Security
 - d. Network and platform interoperability
3. What are the major residual interoperability issues?
4. To what extent has interoperability been provided within product lines, across product lines, through middle–ware solutions, and through standards?
5. To what extent have you found that the following are important in solving interoperability issues?
 - a. Software standardization (within and/or across product lines)
 - b. Platform standardization
 - c. Open system standardization

Software Development Capability Maturity Model

- d. Architectures and testing
 - e. Middle-ware
 - f. Extensible markup language (XML)
6. To what extent do you believe technologies such as Java, common object request broker architecture (CORBA), and component-based architectures will solve interoperability issues? What is the likelihood? What is the time frame? Are there other key technologies?

C. *Managing user expectations*

1. What are the user expectations to be satisfied in your project by the system software product? What is your process for determining the requirements for the software product? What is the acquisition and marketing strategy for the software product to meet your requirements?
2. What are the major quality and interoperability deficiencies you experienced? Can these deficiencies be classified as one of the following?
 - a. Bugs (functions that did not work)
 - b. Functions that did not work as expected
 - c. Functions not included as expected or promised
 - d. Poor and inadequate documentation
 - e. Inadequate training

Software Development Capability Maturity Model

The Software Engineering Institute (SEI) has developed a capability maturity model (CMM) to guide organizations in system software process improvement. This model has contributed to widespread success by helping organizations improve their efficiency in developing quality system software products.

The SEI has defined five levels of CMM as shown in Figure 4–1.

- Level 1 is chaotic.
- Level 2 is repeatable.
- Level 3 is defined.
- Level 4 is managed.
- Level 5 is optimized.

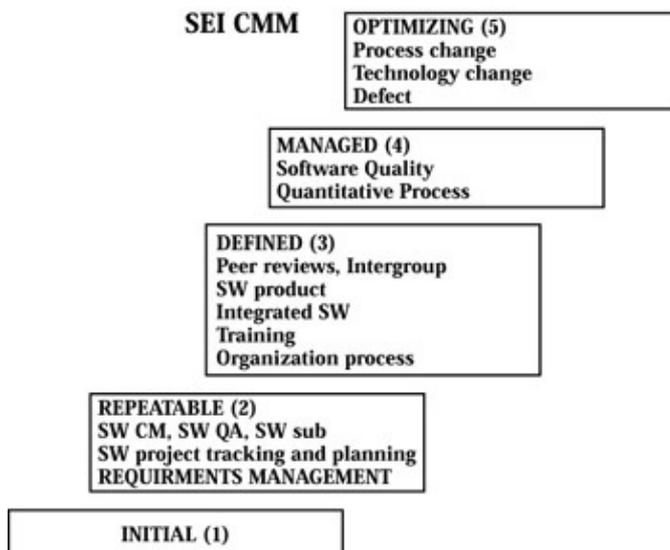


Figure 4–1: Capability maturity model levels

Software Development Capability Maturity Model

The SEI CMM is used as the standard to assess the maturity of an organization's software development process.

Lately, the people capability maturity model (P-CMM) by SEI is a maturity framework that describes the key elements of managing and developing the work force of an organization. Figure 4-2 shows a model of P-CMM. The P-CMM describes that an organization must focus on people, process, and technology to improve performance. The P-CMM includes practices in such areas as work environment, communication, staffing, performance management, training, compensation, competency development, career development, team building, and culture development in an organization.



Figure 4-2: P-CMM model

The success of the system software (SW-CMM) spawned other CMMs that address a wide range of subjects. An SW-CMM provides an organization with a conceptual framework within which specific processes (e.g., configuration management and quality) can be optimized to efficiently improve the capability of the organization. An SW-CMM provides state-of-the-art practices to do the following:

- Determine the maturity of an organization's processes
- Establish goals for process improvement
- Set priorities for immediate process improvement actions
- Plan for a culture of product or service excellence

By focusing on specific processes, an organization can best leverage the resources for their improvement activities while rallying the organization around specific goals. An SW-CMM can be a road map showing an organization how it can systematically move to more mature levels of performance and do it in more effective and efficient ways. After an objective assessment, an organization can set its goals for increasing the capability of its processes.

An SW-CMM can include processes that span the entire life cycle. Starting with requirement management, the processes can span the breadth of product development, ensuring quality, lean production concepts, and support to the field. Each individual process includes elements that provide basic practices and additional practices that add incremental benefits and maturity. When these processes are sufficiently matured, the organization increases its performance or maturity.

Subsequent to the success of the SW-CMM, other CMMs were developed with SEI support. These CMMs included the systems engineering (SE) CMM and the integrated product development (IPD) CMM. It became apparent in the development of these and other models that they all contained common processes (e.g., configuration management, quality, and requirements management), supporting the various types of functional engineering and systems engineering. Improvements in these common processes could benefit other disciplines. Furthermore, it became apparent that process improvement resources applied to one functional discipline (e.g., software engineering) could be beneficial to another functional discipline. The common elements used in an SW-CMM appraisal could be used for SE appraisal, and there would be no need to redo the appraisal of common elements. In addition, improvement efforts based on a unique CMM could

QAS Checklist

result in suboptimization, confusion, and potentially unnecessary expenditure of process improvement resources.

Mark D. Schaeffer of the U.S. Department of Defense says that one of the top-priority projects in the SEI is integration of the CMM products for use in single or multiple functional disciplines. This will greatly enhance the efforts of CMM users and protect the resources already invested. Organizations can use their previous CMM process improvement work and tailor their future efforts to their unique organization. The initial common framework effort will be based on the SW-CMM, the SE-CMM, and the IPD-CMM. Other functional disciplines may be added later. The work accomplished to date in SW-CMM, Version 2.0, and the IPD-CMM have been included in the initial CMM integration (CMMI) baseline. In building these CMMI products, the needs of industry and government partners must be understood and met.

QAS Checklist

The QAS checklist consists of such standards as the International Standards Organization (ISO) and the Institute of Electrical and Electronic Engineering (IEEE) and complies with any other applicable documents:

- Prepare QAS plans, set the tone for quality, and instill quality attitudes among the practitioners.
- Define the purpose and benefits of QAS.
- Explain the elements of a QAS program.
- Explain the procedure steps required for QAS.
- Identify the roles and responsibilities between the project and the QAS.
- Specify QAS reviews, evaluation, and report.
 - ◆ Define quality processes and procedures.
 - ◆ Conduct quality evaluations.
 - ◆ Review products before release.
 - ◆ Maintain quality records and metrics.
 - ◆ Follow up on corrective actions.
- Train the organization in quality processes and procedures.
- Involve evaluation of the project, and attend technical reviews, products reviews, and process reviews.
- Establish a QAS report form, process, policy, and procedure.
- Follow QAS checklists.
- Prepare, submit, and report QA reports.
- Follow the SEI CMM saying: The purpose of QAS is to provide management with appropriate visibility into the process being used by the software project and of the products being built.
- Understand that QAS is an integral part of the IT system development and management process.
- Understand that QAS is a planned and systematic pattern of all actions necessary to provide adequate confidence that the product conforms to established standards.
- Improve quality by appropriately monitoring the system and the development process that produces it.
- Verify full compliance with the established standards and procedures.
- Verify that any variance in the product, process, and standards is brought to the management's attention.
- Review the following major functions of QAS:
 - ◆ Project planning
 - ◆ Requirements process
 - ◆ Design process
 - ◆ Coding practices

QAS Checklist

- ◆ Software integration and testing
- ◆ System integration and testing
- ◆ In-process review of the management and project control process
- Understand that QAS prevents defects:

Early error detection for increased cost savings

- ◆ Assurance for users, customers, stakeholders, and management of product reliability and maintainability
- ◆ Repeatable evaluation process
- Understand that improving quality reduces system development and maintenance costs.
- Plan, perform, check, and act for continual improvement of performance.

Section II: Industry Best Practices

Chapter List

Chapter 5: IT Project Risk Management

Chapter 6: Commercial Best Practices Standards

Chapter 7: System Measurement Techniques

Chapter 8: Commercial Items

IT project managers establish processes to identify, monitor, and manage risks.

Chapter 5: IT Project Risk Management

Project risk is the possibility of unpredictable and unwanted loss that can occur anytime and anywhere during the project's life cycle. Risk is associated with all aspects of the project, such as requirements, architecture, design, implementation, team members, management, and work breakdown structure (WBS). Risk can be due to cost overruns, schedule delays, manpower turnaround, design constraints, influence of new technology, hardware defects, software bugs, communication problems, and network and Internet failure. The project manager continually assesses, identifies, monitors, and manages risk before it becomes a problem. The manager establishes an industry best practices risk management plan to identify and control performance, cost, and schedule risk.

What is Project Risk?

Project risk is a subjective assessment that is made on the probability of *not* achieving a specific objective within the time, costs, and allocated resources. In addition, project risk is the possibility of suffering any loss during the project's life cycle. In a development project, the loss describes the influence to the project in the form of diminished quality of the end product, increased costs, delayed completion, or failure of the project.

A loss associated with a surprise event can take place during the project's system development and maintenance phases. This event creates loss of time, quality, money, control, and understanding among the developers, customers, users, stakeholders, and project manager. For example, if requirements change frequently, the project can suffer from loss of control and understanding. This event can start ripple effects, affecting cost, schedule, and morale of practitioners associated with the project.

Such an event is always likely to affect the project drastically. For example, a project is developed on a host computer to be ported to a target computer when fully tested. Suppose the target computer model is not of the same version as the host computer (i.e., risk probability). The risk probability is measured from 0 (impossible) to 1 (certain). When the risk probability is 1, the risk is called a problem (the probability of failing to achieve a specific outcome and the consequences of failing to achieve that outcome).

The project manager examines each identified risk, refines the description of the risk, isolates the cause, and determines the influence of the risk on the project. The risk influence analysis includes the probability of risk occurrence, its consequences, and its relationship to other risk areas and processes.

Risk Factors

The IT project manager should carefully calculate risk factors. The manager creates and maintains a list of the risk factors at regular intervals until the completion of the project. Risk factors can be due to changes in the project requirements, environment, technology, organization, personnel, and unforeseen events. The three levels of risk factors are low, moderate, and high. The manager prioritizes and resolves high-level risk factors first.

System hazards and safety are related to the system risk factor. A hazard is a set of conditions that can lead to an accident in certain environmental conditions. Safety is defined in terms of hazards. For example, software can contribute to system hazards, and they must be eliminated or controlled to reduce or prevent accidents. Decreasing any or all three of the following risk factors can reduce risk:

1. Likelihood of a hazard occurring
2. Likelihood that the hazard will lead to an accident

Chapter 5: IT Project Risk Management

3. Worst possible potential loss associated with such an accident

During its life cycle, the system can potentially contribute to risk through the effect of the software on system hazards. Therefore the manager should ensure that the software executes within the IT system context without resulting in unacceptable risk. System risk safety involves such factors as identifying hazards, assessing hazards, and controlling hazards risk.

Identification of hazards risk should begin at the earliest stages of system development during concept and requirements definition. The manager should make an initial risk assessment that identifies the safety-critical areas and functions, identifies and evaluates hazards, and identifies the safety design criteria that will be used in the remainder of the system development phases.

Hazard assessment can be viewed as falling along a continuum in terms of severity. One approach establishes a cutoff point on this continuum.

Only the hazards above this point are considered in further system safety procedures. Another approach uses several cutoff points that establish categories of hazards. These categories can be negligible, marginal, serious, or critical when different levels of time and effort are applied.

The manager can introduce hazard control during the system requirements analysis and design techniques to attempt to prevent or minimize the occurrence of a hazard. The objective of system safety requirements analysis is to ensure that the requirements' functionality is specified and implemented and that they are consistent with the safety constraints as shown in Figure 5-1. Ideally, the manager accomplishes this by verifying that the system requirements satisfy the safety constraints and the system correctly implements the requirements.

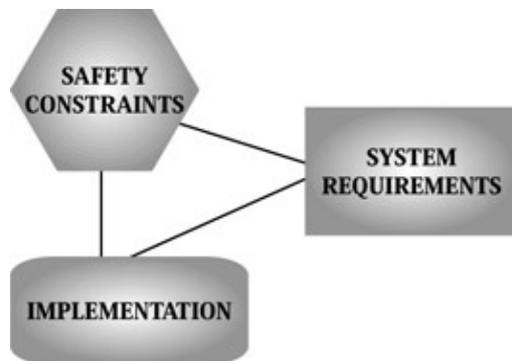


Figure 5-1: Safety risk constraints

An advantage of safety analysis steps is that the errors are caught earlier and thus are easier and less costly to fix. The safety analysis also mitigates the risk factor. The information from early verification activities can help the manager design safety features into code and provide leverage for the final code verification effort. The verification effort is distributed throughout the system (hardware and software) development process instead of being concentrated at the end. Ideally, each step merely requires that newly added detail does not violate the safety verification of the higher-level abstraction at the previous step. Each level is consistent with the safety-related assumptions made in the analysis at the next higher level. These verification activities may have both formal and informal aspects; static analysis uses formal proofs, technical walk-through, and technical reviews. Dynamics analysis involves various types of testing that provide confidence in the models and the assumptions used in the static analysis.

The first step in any safety verification procedure verifies that the system requirements are consistent with or satisfy the safety constraints. This analysis is important so that the code ensures and satisfies that these requirements will be safe. The analysis also identifies important conflicts and tradeoffs early, before the

Risk Management

manager makes design decisions. The manager must make decisions about tradeoffs between safety and reliability or other system and software qualities, and between safety and functionality, for each project on the basis of potential risk, acceptable risk, liability issues, and requirements.

Risk Management

Risk management is a practice of controlling risk. The risk management practice consists of processes, methods, and tools for managing risks in an IT project before they become problems. Figure 5–2 illustrates a risk management process. Risk management consists of actions taken to identify, assess, and eliminate or reduce risk to an acceptable level in such areas as cost, schedule, technical, and products. The risk management process provides a disciplined environment for the IT project manager to make decisions for those areas. The process includes the following factors:

- Continuously assess what could go wrong with risks.
- Determine which risks are important to deal with.
- Implement strategies to deal with those risks.

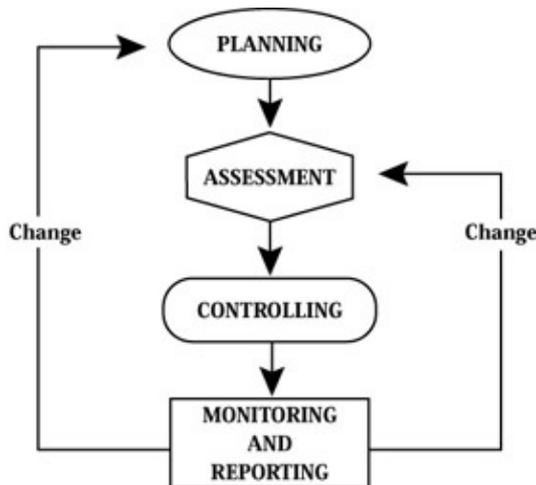


Figure 5–2: Risk management process

In addition, the pressure to reduce costs and improve IT project performance, time to market, and management practices is driving organizations to avoid expensive problems, hence to more effectively manage risk.

The risk management process includes the following four major activities:

1. Risk management plan
2. Risk assessment plan
3. Risk mitigation plan
4. Risk monitoring and reporting plan

Private Risk Management Plan

The risk management plan is an organized, well–structured, comprehensive, and iterative approach for managing risk. The risk management plan records the results of the risk–planning process. The plan includes a set of functions that are identified as continuous activities throughout the life cycle of the project. Each risk

Risk Assessment Plan

goes through these functions sequentially, but the activity occurs continuously and concurrently. Risks are tracked in parallel while new risks are identified and analyzed. A mitigation plan for one risk may yield another risk throughout the project's life cycle.

The risk management plan includes planning, identification, assessment, mitigation, and continuous tracking with control procedures that feed back to the project manager for making decisions as shown in Figure 5–3. Every management action has some type of risk factor involved. A risk management worksheet is a part of the risk management plan and should be designed to assist the project manager in identifying the overall risk level of the phase or project. The worksheet should provide the following:

- Visibility for the subjective factors that may influence the management in the performance
- Identification and measurement of the potential risk areas early in the commitment process
- Identification of the areas that need special emphasis in planning and control
- Estimation of the degree of risk, which should be conveyed to those concerned

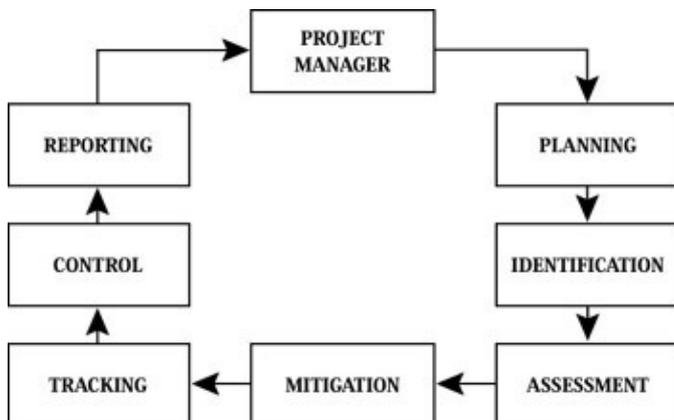


Figure 5–3: Risk management plan activities

The risk management plan includes the following for reducing risk:

- Efficient project management techniques
- Effective communication
- Good control

Larger IT projects (those involving 20 or more practitioners) are at a greater risk of failure than are smaller projects. For example, larger projects need more management involvement, a more structured approach, sufficient decision points and phases, and effective controls. The use of a more structured approach helps the manager review the project after each phase. The larger the project, the greater the need to subdivide it into manageable subprojects, which reduces risk factors. Smaller projects are more manageable because they include more effective management involvement.

Risk Assessment Plan

The risk assessment plan is a process of identifying and analyzing risk and analyzing probability and consequences of identified risks. Risk assessment determines the influence, sensitivity, and relationships of risk and integrates the technical, schedule, and cost assessment. The risk assessment plan starts at the beginning of the project and continues during all phases of the system development and maintenance.

Risk identification is a process of identifying the known and unknown risk events in the project. The location of the risk event can be at various stages in the project. The location can be within WBS and a functional area.

Risk Mitigation Plan

Risk analysis refines the description of an identified risk event. The manager isolates the cause of risk in risk analysis, determines the influence of risk, and determines the type of metrics for high, medium, and low risk as shown in Table 5–1. The manager computes quantitative measures of probability and influence on cost, schedule, and performance.

Table 5–1: Risk Analysis Criteria

High	Moderate	High	High
Probability of occurrence	Low	Moderate	High
Low	Low	Low	Moderate
	Low	Severity of consequences	High

To measure risk is to quantify risk. If the manager cannot measure risk, he or she cannot manage it. The manager establishes a process to measure risk and manage it. The success of an IT project relies on a measure of risk for budget and scheduling, system development and testing phases, and work force.

The ability to quantify risk is essential to the process of budgeting and scheduling. During the process of completing specified tasks, the project manager must be able to verify the estimates and make sound judgments on the risks of cost overruns and time delays. The project manager should discuss the following questions before making sound decisions:

- Do developers with little experience overestimate or underestimate the complexity of the task because of their experience, assumptions they make, models they select, and how they define the model parameters?
- What are the sources of risk associated with project cost estimation? How can such risk be quantified?

Risk measurement process should include the following:

- Constructing the probability density functions
- Probing the sources of risks and uncertainties
- Analyzing and regarding the likelihood of technical and nontechnical risks
- Drawing conclusions on the basis of the accumulated evidence and ultimately selecting the contractors most likely to complete the project without major cost overruns or time delays

Risk Mitigation Plan

The risk mitigation plan determines the project's success or failure. This plan includes various techniques, technology demonstration, prototyping, and test and evaluation. A good mitigation plan includes the following:

- Search for and locate risks before they become problems.
- Transform risk data into decision-making information.
- Evaluate influence, probability, and time frame.
- Classify and prioritize risks.
- Translate risk information into decisions and mitigating actions, both present and future, and implement those actions.
- Monitor risk indicators and mitigation actions.
- Mitigate high and moderate risks.

Risk Mitigation Plan

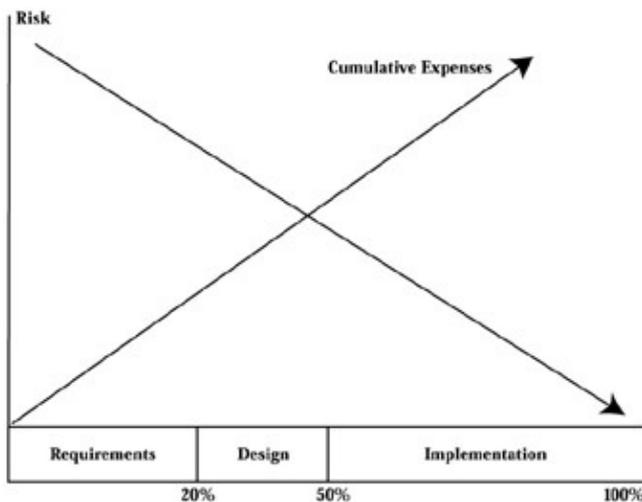
The risks are greater in developing a new IT system than in maintaining or converting an existing system. In new system development, minimization of risks consists of the following:

- Identification of the critical risk elements of a project
- Development of an action plan to anticipate potential problems

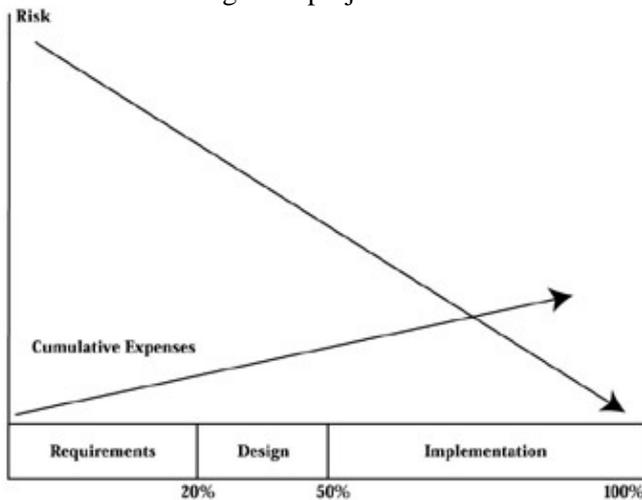
The mitigation plan contains plan-of-action guidelines as follows:

- Emphasize the selection of a methodology.
- Plan the construction of a prototype.
- Provide proper technical training.
- Include tasks and activities.
- Implement the result in each phase.

The risk level decreases as the project moves on in each phase. Figure 5-4, A, shows a well-managed project if the risk and the cumulative cost intersect at the middle during the design phase. An unsuccessful project is illustrated in Figure 5-4, B, which shows that the intersection has gone beyond the implementation phase; this project is at high risk of failure. The project manager should take command and bring the project back to the successful path by identifying, resolving, and controlling risk factors.



5-4 A: A well-managed IT project



5-4 B: A poorly managed IT project

Risk Monitoring and Reporting Plan

In addition, the manager needs to create and implement a continuous process for the effective management of risk. The process includes all interested parties in the project (e.g., developers, customers, users, and stakeholders).

The risk mitigation plan contains metrics as follows to reduce and control risk:

- Planned versus actual computer resources used
- Planned versus actual work force used
- Planned versus actual practitioner turnover
- Number of requirement changes versus original requirements
- System progress showing number of units designed, reused, tested, and integrated
- Cost and schedule (budget versus actual)

Risk Monitoring and Reporting Plan

The risk monitoring and reporting plan consists of systematically tracking and evaluating performance of identified risk areas and events against established metrics. This plan includes cost, schedule, and performance measurement reports. Figure 5–5 shows the risk monitoring process.

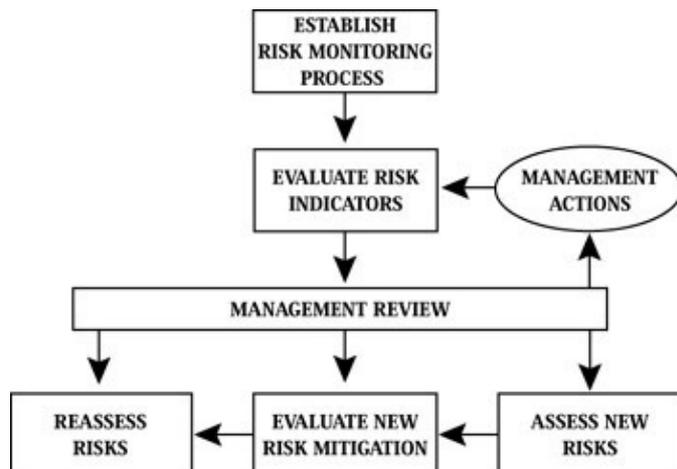


Figure 5–5: Risk–monitoring process

Risk monitoring results and reporting are properly documented for future reference. This document also contains records of risk assessments, risk mitigation, risk analysis, rationale for decisions, and updates.

Principal Best Practices

Successful system software projects use principal best practices. The Airline Software Council identified the following nine principal best practices:

1. Formal risk management
2. Agreement on interfaces
3. Formal inspection
4. Metric–based scheduling and management
5. Binary quality gates at the inch–pebble level
6. Project–wide visibility of progress versus plan
7. Defect tracking against quality targets
8. Configuration management
9. People–aware management accountability

Formal Risk Management

Formal risk management is vital to the success of an IT project. A formal risk management process requires that an organization accept risk as a major consideration for system software project management, commitment of resources, and methods for identifying, monitoring, and managing risk.

The project manager assigns duties to a practitioner as a risk officer and sets up a risk database. The risk officer reports to the project manager if a risk assessment has an influence on the project plan and decision. He or she updates the risk assessment consistent with decision updates during the project. The project manager establishes objective criteria to identify, evaluate, and manage risk. The risk officer verifies that the information flow pattern and reward criteria within the organization support the identification of risk by all project practitioners.

The project manager observes that the integrity of information is properly managed and controlled. The risk officer draws up a risk profile for each risk and regularly updates the risk's probability of occurrence, consequences, severity, and delay. The risk officer confirms that the risk management plan contains explicit provisions to alert the project manager of imminent risks. The project manager encourages all personnel to identify risks and report them to the risk officer.

Agreement on Interfaces

The project manager must complete agreement on interfaces before the implementation phases and maintain such user interfaces as an integral part of the system specification. The Airline Software Council proposed that for those projects developing both hardware and software, the manager must write a separate software specification with an explicit and complete interface description.

The project manager ensures a complete census of input and outputs, which are defined down to the data element level. The interfaces are stable and include hardware, software, users, and major software components. The system specification includes a separate software specification to show the hardware interfaces. The users should verify the interface descriptions as they develop.

Formal Inspection

The formal inspection process identifies and implements the quality of the system products. The project manager integrates the formal inspection process in the project schedule. The manager should conduct formal inspection during all phases of system development and maintenance, including the requirement, architecture, design, implementation, and testing phases.

The project manager establishes procedures, standards, and rules for the conduct of inspections. He or she uses metrics to gauge the effectiveness of inspections. A documented process exists to conduct inspection. The manager properly documents and tracks defects found during inspections and ensures that all deliveries are inspected for quality before they are released for project use.

Metric-Based Scheduling and Management

Metric-based scheduling and management includes statistical quality control of costs and schedule. This requires early calculation of size metrics, projection of costs and schedules from empirical patterns, and tracking of project status through the use of captured result metrics.

Binary Quality Gates at the Inch–Pebble Level

The project manager checks that the cost and schedule performance track against the initial baseline and the latest baseline. The manager tracks the number of changes to the initial cost and schedule baseline and refines the project estimates continuously as the project proceeds. The productivity levels and schedule deadlines are evaluated against past performance and reflected in the risk assessment. The manager monitors the planned versus actual cost and planned versus actual schedule.

Binary Quality Gates at the Inch–Pebble Level

Binary quality gates at the inch–pebble level is the completion of each task, and the phase in the lowest level activity network needs to be defined by an objective binary indication. These completion events should be in the form of gates that assess either the quality of the products produced or the adequacy and completeness of the finished process. Gates may take the form of technical reviews or completion of a specific set of tests, which integrate or qualify system software components, demonstrations, or project audits. The binary indication is meeting a predefined performance standard. The manager can set the standard as a defect density of less than 4% function point. Activities are closed only upon satisfying the standard, with no partial credit given. The manager can apply quality gates at any time during the project.

The project manager checks that the project status and planning estimates have been produced based on the inch–pebble quality gates that can be aggregated at any desirable level. The project manager ensures the following:

- All activities have been decomposed into inch–pebbles.
- All near–term work has been decomposed into tasks no longer than 2 weeks in duration.
- Achievable accomplishment criteria have been identified for each task.
- Tasks are based on overall quality goals and criteria for the project.
- The planned tasks are 100% complete before acceptance.
- All reviews are successfully completed.
- The inch–pebble tasks on the critical path are defined, enabling more accurate assessment of schedule risks and contingency plans.
- The set of binary quality gates is compatible with the WBS.

Project–Wide Visibility of Progress Versus Plan

The project participants should know project–wide visibility of progress versus plan. The project manager establishes an anonymous channel feedback and encourages practitioners to provide bad news up and down the project hierarchy. The project manager ensures that all project personnel know the basic project status. The practitioners can report good and bad as well. The project goals, plans, schedules, and risks are available to the members of the project. The anonymous channel feedback is visible to the project members.

Defect Tracking Against Quality Targets

Defect tracking against quality targets is a process of formally tracking the defects at each phase and activity of the project. Configuration management (CM) enables each defect to be recorded and traced through to removal. The manager compares initial quality targets and calculations of remaining or latent defects with counts of defects removed to track progress during testing activities.

The project manager establishes defect targets for the project and defines consequences if a product fails to meet the targets. The project quality targets apply to all products. The project manager defines circumstances under which quality targets are subject to revision and establishes the techniques that are used to project latent defect counts. The manager also develops techniques to confirm the current projected level of removal defects

Configuration Management

as adequate to achieve planned quality targets. The test coverage is sufficient to indicate that the latent defect level achieved by the end of testing will be lower than the established quality targets.

The project manager checks that the inspection and test techniques employed during the project are effective in meeting quality targets. All discovered defects undergo CM, and the manager performs accurate counts for defects discovered and defects removed. A closed-loop system links defect actions from when defects are first detected to when they are resolved. The defect information is defined at a level of granularity that supports an objective assessment of resolution on a periodic basis.

Configuration Management

CM is a discipline that is vital for the success of an IT project. CM is an integrated process for identifying, documenting, monitoring, evaluating, controlling, and approving all changes made during the life cycle of the system. The CM information is shared by more than one individual or organization.

The project manager integrates the CM process with the project plan. All versions are controlled by CM. The manager uses configuration control tools for status accounting and configuration identification tracking. The manager checks that periodical reviews and audits are in place to assess the effectiveness of the CM process. All information shared by two or more organizations is placed under CM.

People-Aware Management Accountability

People-aware management accountability is a process that the project manager establishes for staffing qualified people and fostering an environment conducive to high morale and low voluntary staff turnover. Qualified people have domain knowledge and similar experience in previously successful projects.

The project manager ensures that domain experts are available for the project. The team members are fully aware of their roles in the project. The opportunities for professional growth are available to all members of the project. The practitioners believe in the goals of the project and that the schedule is feasible. The project manager ensures that the motivation and retention of the team members are key factors in the success of the project.

Risk Management Checklist

Risk can be due to any changes in the requirements or the occurrence of an unforeseen event. The checklist is summarized as follows:

- Risk assessment
 - Risk identification
 - Analysis
 - Decision
 - Risk analysis
 - Identification of risk factors
 - Cost and performance analysis

Configuration Management

Risk prioritization

Computation for risk reduction

- Risk control

Risk reduction

Risk avoidance

Risk transfer

Risk management plan

Process establishment

Preparation of plan to handle risk

Risk resolution

Risk mitigation

Risk monitoring

Risk reporting

Risk evaluation

- Boehm's 10 risk items

Personnel shortfalls

Unrealistic schedules and budgets

Development of the wrong software functions

Development of the wrong user interface

Gold-plating (requirements scrubbing)

Continuing stream of requirements changes

Shortfalls in externally performed tasks

Shortfalls in externally furnished components

Real-time performance shortfall

Straining computer science capabilities

- Head-start factors for risks in an IT project

Significant changes in the organization

Configuration Management

Frequent changes in roles and duties of team members

Change of management during the project's development

Significant changes in requirements

Varying requirements during the project's development

Changes in specifications

Lack of senior management support

Shrinking project budget

Changes in project schedule

High practitioner turnover rate

- Risk management best practices

Establish management reserves for risk resolution

Implement metric-based risk decisions

Perform continuous risk management

Formalize risk tracking and review

Manage influence of external dependencies

Chapter 6: Commercial Best Practices Standards

No single standard is available in the computer industry. Most system developers design their own standards and follow them in their organizations. These standards are variations of the basic IT system development goals and principles. This chapter provides a study of the current commercial best practices standards and tailoring standards guidelines.

Commercial System Development Standards

Commercial system development standards originate from the Institute of Electrical and Electronics Engineers (IEEE) of Europe and Canada and the National Aeronautics and Space Administration (NASA). Almost every sizable computer organization has its own established standard for system development. These standards outline the system development life cycle and the content of required documents. Standards also define system software quality assurance (QA), configuration management (CM), requirement traceability, system design, implementation, testing, and independent verification and validation because they are needed for embedded systems.

System development standards are necessary so that a computer system can interoperate among others. Standards establish uniform system engineering techniques that are applicable throughout the system life cycle. These standards incorporate the best practices that will be cost-effective from a system life-cycle perspective. Standards are intended to be dynamic and responsive to the rapidly evolving system development phases. Data item descriptions (DIDs) that are applicable to the standards are available and provide a set of complete and concise documents that record and communicate information generated from specified requirements.

Characteristics of a Good System Development Standard

A good system development standard is tightly composed and concentrates on system development goals and principles. The standard establishes uniform requirements for system development that are applicable throughout the system life cycle. The requirements of the standard provide the basis for insight into system development, testing, and customer evaluation efforts. Some characteristics of a good standard are listed in Box 6-1.

Box 6-1: Characteristics of a Good System Development Standard

- Has a proper structure that represents clear thinking and a standardized means of communication
- Establishes uniformity
- Follows system development discipline
- Fits in accordance with system development goals and principles and can be used with other standards
- Follows the processes, standards, and procedures (rigorous)
- Provides guidelines, references, road maps, and checkpoints
- Involves quantitative feedback reviews and audits
- Is user friendly and not user hostile
- Includes reasonable documentation
- Provides quality activities and evaluations
- Provides visibility into system development status
- Provides a template model that will introduce a suitable system development method and tools

System Management Standard

- Encourages use of reusable components
 - Encourages tailoring aspects
 - Establishes tests for the system environment
 - Standardizes system development and management processes
 - Provides means that will help develop systems that are efficient and cost-effective throughout a product's life cycle.
-

A good standard defines the requirements for an enterprise's total technical effort related to development of products that include hardware and software and processes that provide life-cycle support for the products. The objectives are to provide high-quality products and services with the desired practitioners and performance features at an affordable price and on time. This involves developing, producing, testing, and supporting an integrated set of products that includes hardware, software, networking, practitioners, data, facilities, and material. The processes include services and techniques that are acceptable to customers, stakeholders, and users.

System Management Standard

The system management standard prescribes an integrated technical approach to engineering of an IT system and requires the application and management of the system's engineering process.

IEEE Standards

IEEE standards provide recommendations that reflect the state-of-the-art system engineering principles for development and maintenance. The IEEE standards meet the requirements of a costly, complex systems' quality, performance, and reliability. The purpose of a standard is to establish and apply requirements during the acquisition, development, and support of software systems. IEEE has designed the following standards for system development:

- P730, Software quality assurance plans
- P828, Standard for software CM
- P829, Standard for software test documentation
- P830, Guide for software requirements specifications
- P1012, Standard for software verification and validation
- P1016, Recommended practices for software design description
- P1058, Standard for software project management plans
- P1061, Standard for software quality metrics methodology
- P1062, Practice for software acquisition
- P1219, Standard for software maintenance
- P1220, Standard for application and management of the systems engineering process
- ISO/IEC 12207, Standard for information technology–software life-cycle processes
- P1233, Guide for system requirements specifications

- P1362, Guide for information technology–system definition–concept of operations document
- P14143, Information technology–software measurement
- P1420, Software reuse
- P1471, Recommended practice for architecture description

Standard for Application and Management of the Systems Engineering Process

The IEEE P1220 standard is for application and management of the systems engineering process. The purpose is to provide a standard for managing a system from initial concept through development, operations, and disposal. This standard defines the interdisciplinary tasks that are required throughout a system's life cycle to transform customer needs, requirements, and constraints into a system solution. The standard guides the development of systems that include humans, computers, and software for commercial and industrial applications. This standard applies to an enterprise within an enterprise that is responsible for developing a product design and establishing the life-cycle infrastructure needed to provide for life-cycle sustainment.

This standard specifies the requirements for the system's engineering process and its application throughout the product's life cycle. The standard does not define the implementation of each system life-cycle process but addresses the issues associated with defining and establishing supportive life-cycle processes early and continuously throughout product development. In addition, the standard does not address the many cultural or quality variables that must be considered for successful product development. The standard focuses on the engineering activities necessary to guide product development while ensuring that the product is properly designed to make it affordable to produce, own, operate, maintain, and eventually dispose of without undue risk to health or the environment.

The P1220 standard describes an integrated approach to product development that represents the total technical effort for the following:

- Understanding the environments and the related conditions in which the product will be used and for which the product must be designed to accommodate
- Defining product requirement in terms of functional and performance requirements, quality factors, usability, producibility, supportability, safety, and environmental influences
- Defining the life-cycle processes for manufacturing, testing, distribution, support, training, and disposal, which are necessary to provide support for products

The standard covers the system engineering management plan, general requirements, the system engineering process, and application of the systems engineering throughout the system life cycle and a list of documentation that will be delivered for maintenance. The general requirements include the following systems engineering processes:

- Requirements analysis and validation
- Functional analysis and verification
- Synthesis
- Design verification
- Systems analysis
- Control

The general requirements also include the following:

- Policies and procedures for systems engineering
- Planning of the technical effort
- Development strategies
- Modeling and prototyping
- Integrated database
- Product and process data package

Standard for IT: Software Life–Cycle Processes

- Specification tree
- Drawing tree
- System breakdown structure (SBS)
- Integration of the systems engineering effort
- Technical reviews
- Quality management
- Product and process improvement

The application of systems engineering throughout the system life cycle includes the following:

- System definition stage
- Preliminary design stage
- Detailed design stage
- Fabrication, assembly, integration, and test stage
- Production and customer support stage
- Simultaneous engineering of products and services of life cycle processes

Standard for IT: Software Life–Cycle Processes

Software is an integral part of IT. The 12207 standard is for the software life cycle. The standard is produced by the International Standards Organization (ISO) and the International Electro–Technical Commission (IEC). The Electronic Industries Association (EIA) also contributed in this standard. This standard provides industry with a basis for software practices. The ISO/IEC 12207 standard is packaged in the following three parts:

- IEEE/EIA 12207.0 is the standard for information technology–software life–cycle processes. This standard contains basic concepts, compliance, life–cycle process objectives, and life–cycle data objectives.
- IEEE/EIA P12207.1 is the guide for ISO/IEC 12207 and called the *standard for information technology–software life–cycle processes–life–cycle data*. This standard provides additional guidance on recording life–cycle data.
- IEEE/EIA P12207.2 is the guide for ISO/IEC 12207 and is called *standard for information technology–software life–cycle processes–Implementation considerations*. This standard provides additions, alternatives, and clarifications to the ISO/IEC 12207's life–cycle processes.

A proliferation of standards, procedures, methods, tools, and environments are available for development and management of software. This proliferation has created difficulties in software management and engineering, especially in integrating products and services. The software discipline needs to migrate from this proliferation to a common framework that can be used by software practitioners to speak the same language to create and manage software. The 12207 standard provides such a common framework. This framework covers the life of software from conceptualization of ideas through retirement and consists of processes for acquiring and supplying software products and services. In addition, the framework provides for controlling and improving these processes.

The highlights of this standard are as follows:

- It is open ended rather than the specification of any particular software development method.
- The software developer is responsible for selecting software development methods and CASE tools that best support the customer's requirements.
- It provides the means for establishing, evaluating, and maintaining quality in software and associated

Standard for IT: Software Life–Cycle Processes

documents.

- The IEEE standard is further modified so that it will be acceptable as an international standard and be a part of the ISO 9000 standard.
- Major features eliminate waterfall, top–down, and hierarchic implications.
- It eliminates the functional area tracks (e.g., software development management, software engineering, etc.), which tend to be waterfall oriented.
- It explains several life–cycle models, tells how an incremental or evolutionary development is planned, and provides guidance for the selection of appropriate deliverables in each increment.
- It eliminates the requirements that partition the computer software configuration item (CSCI) into computer software components (CSCs) and computer software units (CSUs). This requires only a CSCI to be partitioned into components and uses the method that is proposed in the software development plan (SDP) that the software developers create.
- It requires that the software developer lay out a software development process and that it conforms to the life–cycle model, which has been established for the software.
- It acknowledges that each software development activity is an update or refinement of what has gone before rather than a first–time occurrence.
- It reorganizes activities that occur at the same time into activities that are concerned with a given software development activity.
- It explains how each activity is interpreted in the context of builds (e.g., How should the planning activity be divided among builds? How should requirements analysis, design, coding, testing, and product evaluation be performed in incremental builds?).
- It covers in–process reviews and supplements or substitutes for formal reviews.
- It identifies system requirements analysis and system design separately.
- It eliminates the document–driven implications of the ISO 9000 standard.
- The language of the standard is changed from 'software and documentation' to the more generic 'software development products,' which acknowledges nondocument representations, such as data in CASE tools.
- It separates activity requirements from documentation requirements and emphasizes that software development activities need not result in documents.
- It clarifies that deliverable documents are the required outcome of an activity only when so specified on a contract data requirements list (CDRL).
- It is explicit in acknowledging electronic representation of information in lieu of documents.
- It adds guidance about ordering the executable (and possible source) code via contract line item number (CLIN) rather than on a CDRL.
- It adds explicit permission and delivers CASE tool contents for CDRL.
- It clarifies and makes the customer's requirements more consistent for each level of testing.
- It provides clear guidelines for software support.
- It expands and clarifies the requirements, which concern incorporation of reusable software.
- It specifies that nondevelopmental items (possibly modified) will be incorporated into systems under development if they meet user needs and will be cost–effective over the life of the system.
- It interprets this policy for software by itemizing considerations that determine if a reusable software component will meet user needs and be cost–effective over the life of the system.
- It requires that the software developer analyze candidate reusable components in light of these considerations, report the findings and recommendations to the customer, and incorporate reusable components that meet the criteria.
- It specifies allowable substitutions for this standard's required documentation when reusable software is incorporated.
- It makes a preliminary statement about other allowable substitutions (e.g., in testing and formal reviews).
- It defines clearly the interface between software testing and system testing.
- It has a section for planned software QA requirements.

Standard for IT: Software Life–Cycle Processes

- It adds a requirement that will identify, collect, and apply management indicators, and it provides a list of candidate indicators that aid in the selection of a set.
- It revises the requirement on risk management with emphasis on risk as a guiding principle for planning and managing projects.

Box 6–2 contains a list of 22 individual DIDs that are applicable to this standard.

Box 6–2: List of Data Item Descriptions

Plans

- Software development plan (SDP)
- Software installation plan (SIP)
- Software support plan (SSP)

Concept and requirements

- Operational concept document (OCD)
- System/segment specification (SSS)
- Software requirements specification (SRS)
- Interface requirements specification (IRS)

Design

- System/segment design document (SSDD)
- Software design document (SDD)
- Interface design document (IDD)
- Database design document (DBDD)

Test

- Software test plan (STP)
- Software test description (STD)
- Software test report (STR)

User or operator

- Software user's manual (SUM)
- Software input/output manual (SIOM)
- Computer center software operational manual (CCSOM)
- Computer system operator manual (CSOM)

Support

- Version description document (VDD)
 - Software product specification (SPS)
 - Firmware support manual (FSM)
 - Computer instruction set architecture
-

ISO 9000 Standards

A set of six consolidated DIDs combines the DIDs for plans, requirements, design, testing, user or operator manuals, and support; a single DID for small projects summarizes all other DIDs. These DIDs describe a set of documents that record the required information by this standard. The manager should produce deliverable data that use automated techniques.

ISO 9000 Standards

ISO 9000 standards are published by the ISO. The ISO 9000 series consist of the following quality standards:

- ISO 9000
- ISO 9001
- ISO 9002
- ISO 9003
- ISO 9004

ISO 9000 is an overview for selecting the appropriate standard. ISO 9001 covers the 20 elements of an effective quality management system (QMS), which include design, production, servicing, and installation:

1. Management responsibility
2. Quality system
3. Contract review
4. Design control
5. Document and data control
6. Purchasing
7. Control of customer-supplied product
8. Product identification and traceability
9. Process control
10. Inspection and testing
11. Control of inspection measuring and test equipment
12. Inspection and test status
13. Control of a nonconforming product
14. Corrective and preventive action
15. Handling, storage, packaging, preservation, and delivery
16. Control of quality records
17. Internal quality audits
18. Training
19. Servicing
20. Statistical techniques

ISO 9002 addresses all elements of a QMS except design. ISO 9003 focuses on final inspection and testing. ISO 9004 provides implementation guidance and can be used to expand and improve an organization's quality system. If ISO 9004 is in place, an organization can achieve the ISO 9001 standard.

The ISO 9000 series was first published in 1987 and is currently under review. Under the revision, ISO 9002 and ISO 9003 will be incorporated into a new ISO 9001. The new version of ISO 9001 will include the following:

- All types of products and services
- A single standard with flexibility of use
- A more generic approach to conform to different types of industries

Tailoring Standards Techniques

- Alignment with other standards
- Consistent terms, phrases, and definitions

According to James S. Bigelow, a member of the ISO Technical Committee, several boundaries have been identified for the revised ISO 9001:

- Preventing customer dissatisfaction rather than achieving a competitive advantage
- Complying with requirements instead of offering guidance
- Seeking effectiveness over efficiency
- Seeking minimum QA requirements rather than best practices
- Meeting customer requirements rather than enhanced expectations
- Seeking pass/fail instead of degree of performance
- Focusing on improving processes by reducing risks and preventing failures

ISO 9001 is an independent standard that can adapt within reason to meet the needs of an organization (Fellenstein, 1999).

Tailoring Standards Techniques

Tailoring standards for an individual application is essential. Tailoring means the selection of only those products, activities, and reviews that fit the characteristics and are essential to a particular project. The purpose of tailoring helps in the evaluation of requirements in a standard that will save money, prevent duplication, and preserve the schedule of the project. The system developer can recommend tailoring the standards, but ultimately the customer, users, and stakeholders make the final decision.

The result of the tailoring process is reflected in the statement of work (SOW) that prescribes the tasks and reviews. The CDRL contains all of the deliverable documentation. Sometimes clarification is necessary rather than a deletion of requirements. In these cases, the DID for a product can be modified, which will make it unique to the project. When tasks are being added, they may be included in the SOW, which leaves the DID unchanged.

The following are factors that the manager should consider when tailoring a standard:

- System development process that will be used
- System characteristics and intended end use
- Acquisition strategy and type of project management
- Acceptable risk
- Schedule
- Budget
- Development visibility required
- System maintenance concept

Tailoring standards also depends on the class of software. Software classification includes operational, support, system, diagnostic, and automatic test equipment. The tailoring factor changes for the software that is being developed, modified, and reused. Commercial off-the-shelf (COTS) software or nondevelopmental items (NDI) also affect the tailoring factor.

Guidelines for Tailoring

The tailoring process is performed many times throughout the life cycle of a system. Many different events trigger the need for the tailoring of the standards and DID. Tailoring should take place every time the system enters a new phase in the life cycle. Phases include concept exploration, demonstration and validation, full-scale development, and production and deployment. The levels and types of documentation that are needed for each phase vary and should be reflected in the tailoring process. The suggested guidelines for tailoring a standard are as follows:

- Classify the required system by development or nondevelopment category.
- Select activities and reviews in accordance with applicable standards and DID.
- Select deliverable products.
- Tailor the DID.

IT Project Standards Checklist

- Establish project management standards that include the following:

Project progress reporting process

Statement of work–clarification procedures

Project management graphic representations

Staff qualifications and experiences

Cost analysis

Schedule influence

Risk analysis management process

Metrics indicators

Review process and procedure

Tailoring standards plan

Documents deliverable process

Education and training plan

Effective dialog and communication process between system developers and customers, users, and stakeholders

System delivery process

System products acceptance schema

- Establish a system requirements analysis standard that includes the following:

Guidelines for Tailoring

System requirements analysis graphic diagrams

Maintenance of a system requirements list

System external interfaces

Modeling of customer requirements

Requirements feasibility analysis

Requirements traceability schema

Determination of system complexity

System hardware diagram

- Establish a system design standard that includes the following:

System design graphic representations

Allocation of software and hardware requirements

Software development plan

Selection of a suitable method

Selection of a suitable computer-aided software engineering (CASE) tool

Planning for software quality

Planning for increment software development

Planning for software testing

Planning for CM

Planning for corrective action

Reviewing and auditing by the customer

System behavioral design

System architectural design

- Establish a software requirements analysis standard that covers the following:

Software requirements analysis graphic diagrams

Maintenance of software requirements list

Creation of prototyping models

Interface requirements

Guidelines for Tailoring

Establishment of a database

Establishment of a real-time influence

Determination of sizing and timing requirements

Software requirements traceability

Plan for testing requirements

- Establish a software design standard that contains the following:

Architecture design diagrams

Behavioral design diagrams

Object diagrams

Determination of algorithms

Data structure

Data types

Associated operations

Database logical design

Formation of packages, subprograms, functions, and tasks

Functional cohesion

Data coupling

Software design requirements traceability

Compilation dependencies diagrams

Identification, raising, and handling of exceptions

Setting up of a software design file

Software reuse schema

Interfaces

Plan for testing design

- Establish a code style standard that includes the following:

- ◆ Coding formatting

Horizontal spacing

Guidelines for Tailoring

Indentation

Alignment of operators

Alignment of declarations

Blank lines

Paginations

Number of statements per line

Source code line length

◆ Readability

Underscores

Numbers

Capitalization

Abbreviations

◆ Commentary

General comments

File headers

Unit function description

Marker comments

Highlighting

◆ Naming conventions

Names

Type identification

Object identification

Program unit identification

Constants and named numbers

◆ Using types

Declaring types

Enumeration types

Overloading enumeration literals

Guidelines for Tailoring

- ◆ High-level program structure

 - Separate compilation capabilities

 - Subprograms

 - Functions

 - Packages

 - Functional cohesion

 - Data coupling

- ◆ Syntax

 - Loop names

 - Block names

 - Exit statements

 - Naming and statements

- ◆ Parameters lists

 - Formal parameters

 - Named association

 - Default parameters

 - Mode indication

- ◆ Types

 - Derived types

 - Subtypes

 - Anonymous types

 - Private types

 - Data structures

 - Heterogeneous data

 - Nested records

 - Dynamics data

- ◆ Expressions

 - Range values

Guidelines for Tailoring

Array attributes

Parenthesized expressions

Positive forms of logic

Short-circuit forms of the logical operators

Type-qualified expressions and type conversion

Accuracy of operations with real operands

◆ Statements

Nesting

Slices

Case statements

Loops

Exit statements

Safe statements

'Go to' statements

Return statements

Blocks

◆ Visibility

Use clause

Rename clause

Overloaded subprograms

Overload operators

◆ Exceptions

Disasters versus state information

User-defined, implementation-defined, and predefined exceptions

Handlers for others

Propagation

Localization of the cause of an exception

◆ Erroneous execution

Guidelines for Tailoring

Unchecked conversion

Unchecked deallocation

Dependence on parameters–passing mechanism

Multiple address clauses

Suppression of exception check

Initialization

◆ Tasking

Tasks

Task types

Dynamic tasks

Priorities

Delay statements

◆ Communication

Defensive task communication

Attributes count, collable, and terminated

Shared variables

Tentative rendezvous constructs

Communication complexity

◆ Termination

Normal termination

Abort statement

Programmed termination

Abnormal termination

Chapter 7: System Measurement Techniques

Measurement is a key element in the management of successful IT projects in every established engineering discipline. The objective of measurement is to provide IT project managers with the system information required to make informed decisions that influence project cost, schedule, and technical objectives. This chapter covers project measurement methods and tools.

Project Measurement

Project measurement is a discipline that helps IT projects stay on schedule. Project measurement is a management tool that the project manager uses to identify risks, track specific problems, and assess the influence of these problems on project cost, schedule, and performance objectives. Project measurement provides accurate data to help the manager monitor the project's progress against the plan. The benefits of IT project measurement are as follows:

- **Enhances effective communication.** Communication among the members of the project is vital for its success. A well-known saying applies to communication in the IT industry: 'In the kingdom of the blind, the one-eyed man is king.' It is good practice to establish project measurement and share the information among the members of the team to avoid such an environment. This process establishes effective communication throughout the project and reduces the ambiguity that often surrounds many issues on a project. Measurement allows such issues to be explicitly identified, prioritized, tracked, and communicated at all levels of the project.
- **Corrects problems.** Correction of problems early in the project saves time and cost. Measurement focuses attention on the early discovery and correction of technical and management problems that can be more difficult to address later. The manager identifies potential problems early as risks that should be assessed and managed.
- **Assists in tradeoffs.** The manager decides tradeoffs on the basis of the project measurement data. Cost, schedule, quality, functionality, and performance all have to be managed together to make the project a success. A decision in one area often has an influence on decisions in other areas. Measurement allows the project manager to objectively assess these influences and make the proper tradeoff decisions to best meet project objectives. Even in highly constrained project environments, measurement helps the manager optimize performance within the scope of project objectives.
- **Tracks project activities.** Project tracking accurately describes the status of the project and represents the progress of the project activities. It helps the manager answer key questions, such as 'Is the project on schedule?' and 'Is the project product ready to deliver?'
- **Helps managers make correct decisions.** Making decisions is the primary duty of an IT project manager. The more accurate the measurement, the more accurate the data and the more correct the project management decisions. Measurement provides an effective rationale for selecting the best alternative.

Measurement establishes a basis for objective communication within the project team. This process helps the manager make decisions that materially influence the outcome of a project quickly and correctly. Measurement provides a baseline quantitative process for implementing risk management and financial performance on a project (Figure 7-1).

Project Measurement Metrics

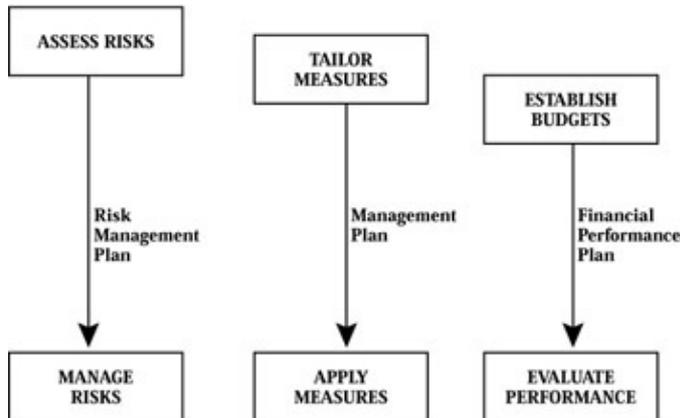


Figure 7–1: Quantitative project measurement process

Project Measurement Metrics

Project measurement metrics is an excellent method of keeping track of the project's progress, various activities, performance, and expenditures. Metrics have been practiced in the IT industry for many years. In fact, people use measurements and metrics almost every day (e.g., driving a car, budgeting earnings and expenses, and preparing a list of necessities before going to the market).

If the manager cannot measure the progress of a project, he or she cannot manage it successfully. A successful project has the following factors:

- Goals
- Time
- Milestones
- Cost
- Budget
- Testing

A successful project 5 Plans 1 Budget 1 Schedule 1 Manpower. Project measurement plans consist of metrics that keep track of the project's progress (Figure 7–2). The following are characteristics of these metrics:

- Metrics without management discipline are not accurate metrics.
- A major factor of a successful project is an appropriate budget. Metrics showing time and cost accurately reflect budget.
- The project budget is directly correlated to the requirements. The project manager estimates project cost after analyzing and understanding the project requirements.
- Cost metrics help the manager spot problem areas.
- Cost saving is time divided by milestones. Each milestone defines tasks to be completed by that point. To keep projects on schedule, managers trade requirements for time.

Case Study 7-1

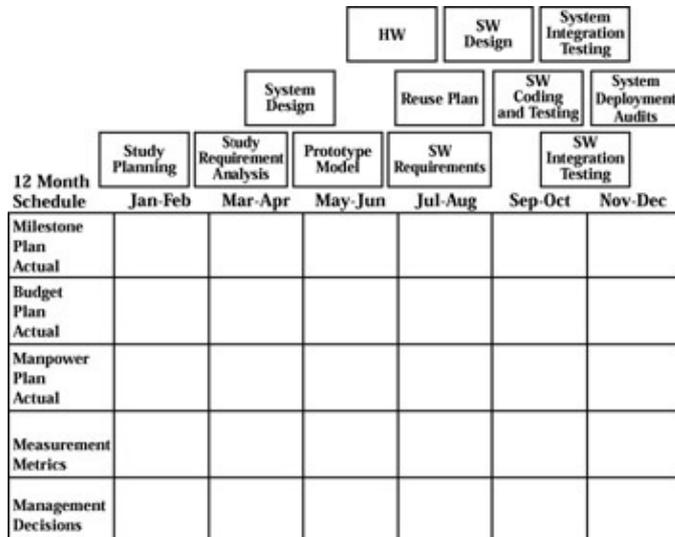


Figure 7-2: Project measurement process

The manager calculates typical metrics for the progress of manpower, hardware defects, software bugs, and travel and training expenses. A project without testing will be unsuccessful. Testing metrics help the manager verify and validate requirements and the correctness of the product. Testing metrics provide the manager with continuous feedback for the correctness of the project. Monitoring the project's progress leads to corrective action. The project manager must measure and pay attention to what the data are revealing. Case Study 7-1 provides a typical measurement example.

Case Study 7-1

In the early 1990s, Debra Domeyer inherited a financial application development project in which the project team was extremely busy accomplishing little. The project, which was at a mortgage consolidation company, was rudderless, lacking a solid project plan, an authoritative project manager, and good tracking measures. More than a dozen people were at the first planning meeting. Domeyer attended, but there was no clear decision maker, and the project had a marathon scope.

'It had become a financial system that would also brush your teeth and make your bed in the morning,' says Domeyer, who is now Chief Information Officer (CIO) of PG&E Energy Services in San Francisco, California.

After the fateful first meeting, Domeyer estimated that the project was only 5% complete after 5 months of work. Given the anticipated spending rate and delivery speed, the new financial system would cost more than \$1 million and take at least 2 years to complete. She reviewed the numbers with her Chief Financial Officer (CFO) and other senior executives, who quickly agreed to cancel the project. Domeyer then launched a new project with strict time, cost, and function requirements. She installed upgrades to the existing financial software package and met the most pressing business needs in 4 months—all for roughly \$140,000.

A good project manager is the most important tool for successful delivery of a complex IT project, but even the most knowledgeable manager needs more than experience and gut feelings to close intricate IT implementation projects successfully. The manager can use a metrics discipline of instituting measurement programs to keep projects on schedule. Metrics can track everything from programmer productivity to business value.

Management Measurement Criteria

Although it seems like common sense to measure a complex IT project's progress systematically, most CIOs bury metrics beneath all of the other demands on their time. In a 1999 study of IT managers in 6000 companies spanning major industries in 28 nations, Rubin Systems Inc., a consultant in Pound Ridge, New York, found that only 50% of the respondents actively use metrics. Of these, only 4% collect and use them on a weekly basis, 10% use them monthly, and 1% use them once or twice a year (Radosevich, 1999a).

Management Measurement Criteria

The following are management measurement criteria for the success of an IT project:

- Schedule and progress regarding work completion: cost, schedule, and development progress (Figure 7-3)

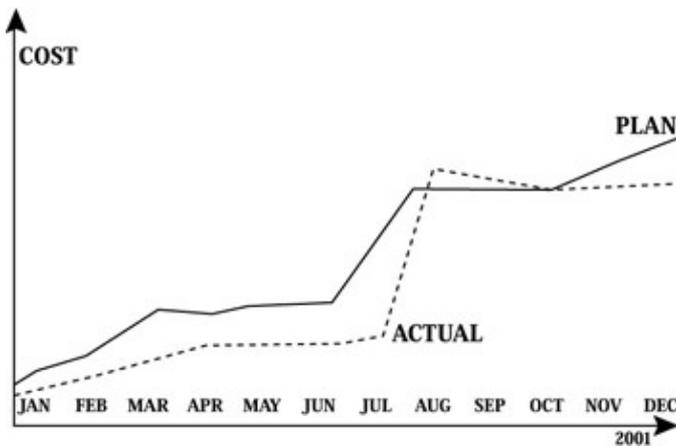
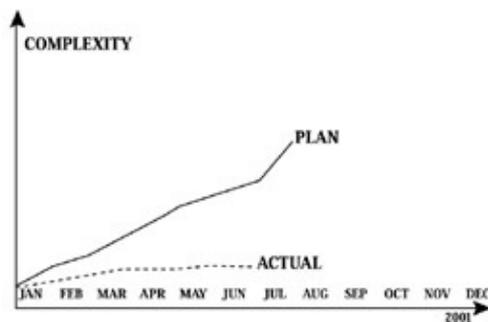


Figure 7-3: Sample cost, schedule, and project progress

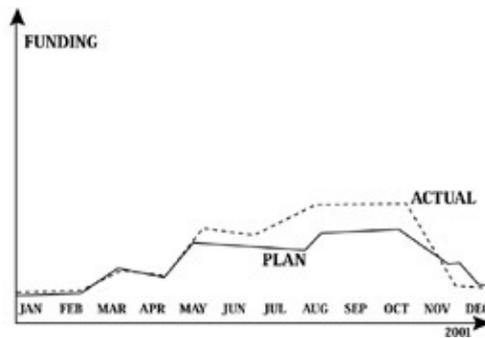
- Growth and stability regarding delivery of the required capability: requirement traceability, requirement stability, design stability, development progress, and complexity (Figure 7-4, A)



7-4 A: Sample cost, schedule, and project progress

- Funding and personnel resources regarding the work to be performed: cost and manpower (Figure 7-4, B)

Best Practices System Metrics



7-4 B: Sample funding and personnel resources

- Software development performance regarding the capabilities to meet program needs: software engineering environment
- Technical adequacy regarding software reuse, language, and use of standard data elements: computer resource use

The purpose for each of these metrics is as follows:

1. **Cost.** Tracks software expenditures (dollars spent versus dollars allocated)
2. **Schedule.** Tracks progress versus schedule (event/deliverable progress)
3. **Computer resource use.** Tracks planned resource capacity versus actual size (percentage of resource capacity used)
4. **Software engineering environment.** Rates developer's environment (developer's resources and software development process maturity)
5. **Manpower.** Indicates the developer's application of human resources to the development program and the developer's ability to maintain sufficient staffing to complete the project
6. **Development progress.** Indicates the degree of completeness of the software development effort; can also be used to judge readiness to proceed to the next stage of software development
7. **Requirement traceability.** Tracks requirements to code (percentage of requirements traced to design, code, and test cases)
8. **Requirement stability.** Tracks changes to requirements (user and developer requirement changes and their effects)
9. **Complexity.** Assesses code quality
10. **Breadth of testing.** Tracks testing of requirements (percentage of functions and requirements demonstrated)
11. **Depth of testing.** Tracks testing of code (degree of testing)
12. **Fault profiles.** Tracks open versus closed anomalies (total faults, total number of faults resolved, and amount of time faults are open by priority)
13. **Reliability.** Monitors potential downtime (software's contribution to mission failure)
14. **Design stability.** Tracks design changes and effects (changes to design, percentage of design completion)
15. **Product quality.** Regards the delivered products (fault profiles, reliability, breadth of testing, complexity, and depth of testing)

Best Practices System Metrics

Best practices system metrics are calculated or composite indicators based on two or more measures. These metrics are used for measuring the progress of a project, improving a process, selecting methods and tools, and improving communication among practitioners. Comparison of this metric with a previous project provides a baseline for the data and gives meaning to the metric. Case Study 7-2 provides a typical metrics

example.

Case Study 7–2

In 1997, when Jim Harding joined Olsten Corporation as a CIO, the \$4.6 billion staffing and health services company had suffered several false starts trying to implement a nationwide, integrated IT system. The abandoned projects had cost roughly \$5 million, years of wasted time, and the information system (IS) department's credibility.

Harding was not taking chances. He knew he had to put detailed measurement programs in place to keep the lid on the \$20 million project. The company's existing systems traced their origins back to 1969, when Olsten was a \$100 million company and a single mainframe held sway over the data center. Since then Olsten has added a few IBM AS/400 computers to service divisional office support functions but not much else. Harding would have to start from scratch, adding all new PC–based local area networks (LANs), remote e–mail, enterprise resource planning systems, and a data warehouse in 30 months.

By 1997, Olsten was attracting major customers that wanted national and global contracts with such features as discounts, customized reporting, and consolidated invoicing. Harding's task was to provide systems that would let Olsten better manage its own burgeoning base of 80,000 contract workers and provide customers with global account management.

To keep his renewal project from imploding, Harding developed a project metrics program that tracked the following:

- **Milestone.** Timing the major chunks of the project, such as the LAN installation; project team members tracked hundreds of subevents leading to each milestone
- **Performance testing.** Simulating the heavy demand that Olsten's employees would put on the applications and networks
- **Data conversion.** Identifying data that had to be transferred to the new systems and writing programs to convert old to new
- **Cost.** Tracking expenses against original budget projections

The measurement program alone cost roughly \$2 million, or 10% of the total cost of the project. However, with the company's entire IS system hanging in the balance, the metrics illuminated the difference between victory and ignoble defeat.

The project had its ups and downs, and the metrics played a role in both. For instance, with roughly 2 months to go before the company's new financial system was set to begin, Harding's data conversion spreadsheet showed that his team would need 2 weeks to load the new data, bringing the system down while they did it. The project team squeezed the conversion time down to 4 days by working with business leaders to purge old data and reduce the data they needed to convert and by tuning the financial application software. 'If we had not been able to reduce [the conversion time], we would not have been able to implement,' Harding says.

Achieving the milestones led to success. For instance, after installing the PC–based LANs, Olsten employees had a company–wide Intranet and e–mail system for the first time.

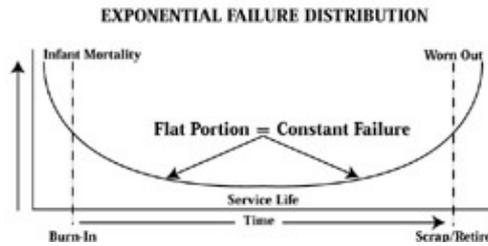
Olsten announced successful completion of its new system in June of 1999. Based on his experience, Harding advises devising metrics to catch mistakes and create smaller victories along the way. 'Then end users stick with you when the inevitable bumps arise,' he says (Radosevich, 1999b).

Project Software Measurement

Project software measurement is an integral part of a successful IT project. A system consists of hardware and software. Hardware can be seen and touched. Software cannot be seen nor touched, but it is essential to the success of a computer system. The measurement process of software emphasizes software error prevention, fault detection, and removal. The measurements depend on project constraints such as resources, schedule, and performance.

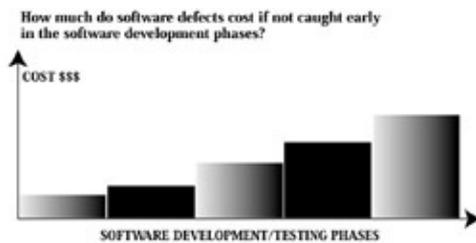
A software error is usually a programmer action or omission that results in a fault. A fault is a software defect that causes a failure, and a failure is the unacceptable departure of a program operation from program requirements.

A difference exists between hardware failure rate (Figure 7–5, A) and software failure rate. When the hardware component is first manufactured, the initial number of faults is high but then decreases as the faulty components are identified and removed or the components stabilize. The component then enters the useful life phase, where few if any faults are found. As the component physically wears out, the fault rate starts to increase.



7–5 A: Bathhtub curve (hardware defect);

Software, however, has a different fault or error identification rate. The error rate is at the highest level at integration and test. As the software is tested, errors are identified and removed (Figure 7–5, B). This removal continues at a slower rate during its operational use, the number of errors continually decreasing assuming that no new errors are introduced. Software does not have moving parts and does not physically wear out as does hardware, but it does outlive its usefulness and become obsolete.



7–5 B: Software defect

The focus concerning software errors must be on comprehensive requirements and a comprehensive testing plan. The IT project manager must do the following:

- Start with the requirements, ensuring that the product developed is the one specified and that all requirements clearly and accurately specify the final product functionality.
- Ensure that the code can easily support sustaining engineering without infusing additional errors.
- A comprehensive test program is included that verifies all functionality stated in the requirements.

Tailoring Project Measurement

Tailoring project measurement is a process (Figure 7–6). The objective of the measurement tailoring process is to define the measures that provide the greatest insight into project issues at the lowest cost. Issues are real or potential obstacles to the achievement of project objectives. The tailoring process is composed of the following three activities:

1. **Identification and prioritization of project-specific issues.** The IT project manager derives issues from project–context information, management experience, and risk–assessment results. The manager assigns priorities to each issue to establish its relative importance as a factor in selecting appropriate measures.
2. **Selection of appropriate measures to address the project-specific issues.** The selection activity employs a defined framework that maps common project issues to measurement categories to measures. These selections result in measurement data requirements that can be incorporated into the project book.
3. **Integration of the measures into the developer's software process.** The project manager must consider the suitability of the selected measures in the context of the developer's software process and overall technical approach. The manager should use measurement requirements not to change the developer's software process but to gain insight into it. When implementing measurement on an existing project, the manager should give special consideration to existing data sources and ongoing measurement activities. The manager documents the result of the tailoring process in a project measurement plan.

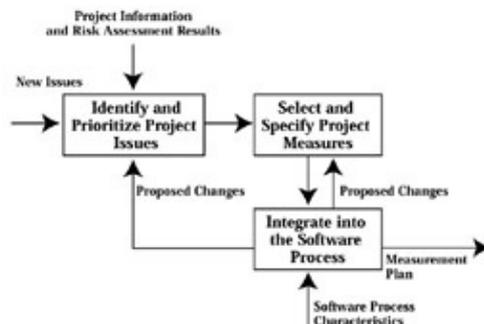


Figure 7–6: Tailoring process

Practical Software Measurement

Practical software measurement (PSM) provides the IT project manager with the objective information needed to successfully meet cost, schedule, and technical objectives on software–intensive projects. The project issues drive PSM. The PSM describes software measurement as a systematic but flexible process that is an integral part of the overall project management structure. PSM is based on actual software measurement experience on various projects. PSM treats measurement as a flexible process, not a predefined list of graphs or reports. The process is adapted to address the specific software issues, objectives, and information requirements unique to each project. The PSM measurement process includes the following set of industry best practices (Roedler, 1999):

- **Schedule.** The schedule measurement includes work unit progress and milestone performance. The work unit progress covers the status of the problem report, management tracking, requirements, various phases of the life cycle, technical reviews completed, and change requests. The milestone performance covers dates, schedule dependencies, lead time, slack time, and the critical path method

Project Measurement Checklist

(CPM).

- **Cost.** The cost measurement includes personnel, financial, and resource measurements. The personnel measurement covers effort and staffing. The financial measurement covers cost and earned value (EV). The resource measurement covers quantity, availability, and use.
- **Product.** The product measurement covers functional and product size and stability. Functional size and stability contain requirements; system functions; requirements added, deleted, and changed; and requirement traceability forward and backward. Product size and stability contain interfaces and database size.
- **Quality.** The quality measurement includes functional correctness, efficiency, reliability, usability, maintainability, and portability. Functional correctness covers problem reports and defects. Efficiency covers throughput, use, and time. Reliability covers system failures. Usability covers learning difficulty, whether the system is user hostile or user friendly, operational errors, and customization difficulty. Maintainability covers maintenance time and update feasibility.
- **Performance.** The performance measurement includes evaluation of efficiency, effectiveness, and updates. The evaluation process covers capability and audit. The efficiency evaluation process covers productivity and cycle time. The effectiveness process covers completion of system development and maintenance phases. The update process covers the size and effort of rework.
- **Technology influence.** The technology influence measurement includes maturity and effectiveness of technology. The technology maturity includes the stability of technology and its adequacy for the IT system. The technology influence covers its implementation and functionality for the specific system.
- **Customer.** The customer measurement covers satisfaction with the system and feedback. The customer feedback includes award fee amounts, survey results, and number of commendations and complaints.

PSM integrates the measurement requirements into the software developer's process. The project manager tailors the measurement set for each project to ensure that the measurement process is cost-effective and that measures provide meaningful and usable results.

PSM defines an issue-driven analysis approach, which helps the project manager make informed software decisions. The PSM analysis approach incorporates the use of multiple measures and nonquantitative program data to identify and evaluate software problems.

PSM defines a nonprescriptive measurement and provides a mechanism for the objective communications necessary within an integrated product team (IPT).

PSM provides a basis for enterprise-level software management. PSM is designed to help the manager put measurement into practice at the project level, thereby providing the data required to address enterprise-level software performance, process improvement, and business-related questions. PSM also supports IT performance measurement requirements.

Project Measurement Checklist

- Ensure that everyone in the project understands the capabilities and limitations of the measurement process.
- Start small. Implement only a few measures to address key issues, and show how the measurement results support both the project and higher-level management objectives.
- Do not allow anyone in the project to use measurement to evaluate individual or work-group performance.
- Ensure that only the required measures are implemented based on the issues and objectives of the project. Do not collect data that are not needed. The measurement process must be cost-effective to

Project Measurement Checklist

succeed.

- Make the measurement data and information available to everyone on the project.
- Initially implement the measurement process with basic, commercially available database, spreadsheet, word processing, and presentation graphics applications.
- Make measurement an integral part of the project.
- Ensure that all users at all levels understand what the measurement data represent. This understanding is vital to the proper interpretation of the measurement analysis results.

Chapter 8: Commercial Items

Commercial items (CIs) consist of automated software tools for management of IT projects. CI tools consist of commercial–off–the–shelf (COTS) and nondevelopment items (NDIs) that are already developed in an organization. The CI tool encompasses a collection of automated software tools that assist IT project managers in the various phases of system management. The CI tool is not a replacement for any method of the system management; it is a supplement for the methods and enhancement for generation of quality products.

Various CI tools are available for project management, system engineering, system architecture, system requirements analysis, system design, software requirements analysis and design, rapid prototyping, configuration management (CM), software quality assurance (SQA), software complexity measurement, testing, and documentation preparation. This chapter discusses only project management–related CI tools.

Definition of CI Tools

Vendors develop CI tools at their own expense for the industry and sell them in substantial quantities on a competitive basis to multiple organizations. CI tools also consist of NDIs that have been developed in an organization and are available for reuse. CI tools apply rigorous discipline in the management of an IT project. The manager must find the right CI tools for the right management applications and environments. The practitioners who use the tools must be properly trained. This approach saves time and cost of project management and achieves efficiency and quality in the final product. The following are benefits for use of CI tools:

1. Save costs
2. Reduce schedules
3. Allow for faster project development
4. Allow for reuse of state–of–the–art practices
5. Enhance quality
6. Improve testing

CI tools are only devices that aid the work of project management; the tools by themselves have little value. The effectiveness of CI tools depends on the managers who are using them and the quality of the tools. Thus a CI tool can be defined as a computer program that aids in the management, development, analysis, testing, or maintenance of another computer program or its documentation.

CI Selection Criteria

CI selection criteria depend on a particular requirement, environment, and ideas about how the tools should work. The project manager selects a CI tool suitable for the magnitude and complexity of the project that he or she manages and for the number of resources involved in the project. The selection depends on the project schedule, budget, and complexity of interdepartmental interfaces. No industry standard exists for evaluation of a CI tool.

The selection of a CI tool depends on the project manager's experience with project management CI tools. The following are six guidelines for evaluation of a CI tool:

1. Ease of use
2. Capability
3. Robustness

4. Functionality
5. Ease of insertion
6. Quality of support

Ease of Use

Ease of use is a measure of a CI tool's effectiveness for interaction by a user. The functionality or completeness of a tool does not matter. If the user spends most of his or her time thinking about how the tool works, then the tool is a hindrance and not helping with the task.

A CI tool should be user friendly and not user hostile. The tool should be easy to use and capable of tailoring to the needs of a particular user. The tool should be helpful to the user and perform particular functions. The more intelligent a tool is, the more functions it will perform without direct specification by the user. The tool should anticipate user interaction and provide a simple and efficient means for execution of functions.

Unpredicted responses from the CI tool usually lead to unhappy users and unwanted output. Command names should have a function. The user should rarely be surprised by a tool's response. The tool should check the user's errors and check and correct these errors whenever possible. A good tool will accommodate interaction with many users or other tools.

Capability

The capability of a CI tool depends on how well it understands the product that it is manipulating and that simple commands can cause major effects. The tool also demonstrates its capability by reasonable performance, which is achieved with efficient use of management resources. A good tool can give the impression of greater power capability, which keeps more knowledge about its internal state.

The performance of a CI tool greatly affects the ease with which it is used and can ultimately determine the success of a tool within an organization.

Robustness

The robustness of a CI tool is a combination of the following factors:

- Reliability of the tool
- Performance of the tool under failure conditions
- Criticality of the consequences of tool failures
- How the tool is integrated into the environment

The consistency of the tool's operation confirms its degree of robustness. Consistency relates to well-defined syntax and semantics. A CI tool evolves over time and accommodates the changing requirements, changing environment, correction of detected flaws, and performance environment. A good tool should be built, evolve, and retain compatibility between versions. A CI tool is a piece of software that performs a function and may not be free of bugs. A tool should be self-instrumented and assist in determining the cause of a problem. The tool should contain a self-test mechanism that ensures that it is working properly.

Functionality

The functionality of a CI tool is driven by the task for which it is designed and by the methods used in the accomplishment of that task. Many tools are available in the IT industry. The accuracy and efficiency with which the tool supports a method can directly affect the understandability and performance of the tool and determine the quality and usefulness of tool outputs.

The tool should provide an adequate scheme for storage, organization, and manipulation of the products of the project's application. The tool should also provide guidance and ensure that the user follows the concepts of the methodology.

During the CI tool selection process, the manager should establish and observe guidelines so that the tool operates correctly and produces the correct output. The tool should generate output that is consistent with what is dictated by the methodology.

Ease of Insertion

Ease of insertion is the ease with which a tool can be incorporated into the target environment. Managers and users need to be aware of how well the tool fits within the existing environment, and they must accept changes that the tool may inflict on their environment. When selecting a tool, the project manager should know how much effort is necessary to effectively learn the tool and put it into practice. The tool's command set should be consistent and understandable. The tool interacts with the user and helps the user learn proper use of the tool. The tool should have templates or other aids that guide interaction.

A good CI tool should run on the existing system and be easy to install and learn. A good tool should use file structures and databases that are similar to what is currently used. The data should be able to interchange between the tool and other tools that the organization currently employs.

Quality of Support

Quality of support refers to the ranges from cost–maintenance agreements to the level of required training provided. When selecting a tool, the project manager should consider the past performance of the vendor and the product. The tool should be sound and mature, and the product should be from a well–reputed vendor. The purchase or rental agreement should answer the following questions:

- What is or is not being required?
- Is there a cost reduction for the purchase of multiple copies?
- Is a corporate site license available?
- Can the tool be leased?
- Are there warranty periods in which a CI tool can be returned for a full refund?
- Can a customer acquire full rights and access the source code?
- Is the user free of all vendors' obligations?
- Is a maintenance agreement available?
- Can the user receive future updates free of charge?
- Are the future updates compatible with the existing version?
- Does the vendor provide a responsive helpful hotline service, and what is the turn–around time for problem reports?
- What is the delivery time of the tool?
- Does the vendor have an effective training program available? Are examples and exercises available for self–study courses? How much free training is included with the purchase of the CI tool?

Microsoft Project 98

- Are vendor representatives knowledgeable and well trained?

The CI tool vendor should support it with ample documentation manuals for installation, use, maintenance, and interfacing. Documentation provides a big picture of the tool and what it does. The documentation should be simple to read, understandable, complete, accurate, and affordable.

Project management tools relate best to the IT project manager's work. They are software packages that center on a database engine, a place where the project data are stored and manipulated. Many CI tools are available for project management, some of which are described as follows:

- Microsoft Project 98
- Kidasa Software's Milestones Etc. 5.0
- AEC Software's FastTrack Schedule 6.0
- Digital Tools, Inc., AutoPlan
- Computer Associates International, CA-SuperProject for PC
- COSMIC, COMPASS
- Quality Software Products Co., MasterPlan
- Cambridge Management Systems, Project Outlook

Microsoft Project 98

Microsoft Project 98 is easy to install but requires a reboot to finish installation. Project 98's initial data entry is in the timeline format known as a Gantt chart, which calls for assignment of start and finish dates and calculation of the time allotted to each task. New lines are inserted above the task. Project 98 indents tasks to denote hierarchy. Dragging a task's timeline bar and connecting it to another can link subtasks. Editing the project data is not comfortable. As each task and subtask is added or edited, associated time lines disappear and the task reverts back to the default—the current day's date. The user must enter all previously scheduled information.

Project 98 reclassifies the subtasks as a main task when a main task is deleted. When the user moves the end point of an independent task, the start points for dependent tasks cascade to correspond with the new end point. Spell check is available.

A project manager needs to excel in presentation, diagramming the entire project in an understandable way. Project 98 helps the project manager make more comprehensive views, including Gantt and resource graphs, variations on resource usage tracking graphs with Gantt formats, and a calendar view.

Project 98 can easily publish to the Web. The user can save a file as a hypertext markup language (HTML) document and save schedules in a graphic format.

Project 98 has many advanced features, such as open database connectivity support for transferring files to various databases. The tool also tracks resource allocation, workload, and labor costs as an integrated function. Project 98 gives standard telephone support with a charge for after-hours calls, and it has a Web interface for posting questions online. Project 98 includes the standard help topics, three tutorials, and a series of links to Microsoft web sites. Project 98 has links to frequently asked questions (FAQ) lists and an online support form. The major characteristics of Project 98 are as follows (Microsoft Corp., Redmond, Washington, 425-882-8080, www.microsoft.com/project):

- Hardware requirements include Windows 9x or NT, 32 Mb RAM, 40 Mb storage, and 133 MHz Pentium.
- Ease of installation is good.

Kidasa Software's Milestones Etc. 5.0

- Data entry is excellent.
 - ◆ Intuitive data entry format: Yes
 - ◆ Data hierarchy: Yes
 - ◆ Dependent tasks connected to independent ones: Yes
- Data editing is fair.
 - ◆ Easy to go back and edit: No
 - ◆ Dependent data remain after removing data with a dependent task: Yes
 - ◆ Cascading time lines: Yes
 - ◆ Spell check: Yes
- Presentation is excellent.
 - ◆ Charts easy to make: Yes
 - ◆ Gantt timeline: Yes
 - ◆ Calendar: Yes
 - ◆ Program evaluation and review technique (PERT) charting: Yes
 - ◆ Resource allocation: Yes
 - ◆ HTML file creating: Yes
 - ◆ Color publishing: Yes
 - ◆ Graphics creation from scheduling: Limited
 - ◆ JPEG: No
 - ◆ Bitmap: No
 - ◆ GIF: Yes
 - ◆ TIFF: No
- Advanced features are excellent.
 - ◆ Object linking and embedding (OLE) available: Yes
 - ◆ Workload tracking: Yes
 - ◆ Cost and hour tracking: Yes
- Help information is fair.
 - ◆ Telephone support: Limited, some charges apply
 - ◆ Online help: Extensive, tutorials and help
 - ◆ Web site: Form to submit questions
 - ◆ Hardcopy manual: Yes

Kidasa Software's Milestones Etc. 5.0

Milestones Etc. focuses on drawing time lines and entering explanatory text. The tools have visual cues to show progress and status. The tool is easy to install. Milestones asks only one installation question: standard or custom? Milestones sets up a separate text box for subactivities that the user can drag and drop onto the timeline. Symbols and connectors make the timeline bars represent similar tasks. The user can insert new tasks by right-clicking on a task to create a new line below.

Milestones lists subtasks under a task but does not indent them. The user can edit Milestones text by clicking on the task's date to open a new window for data. A mouse click on the time makes another data point instead of editing one. The user cannot edit the connecting bar colors, so the user must delete and recreate the timeline bar in the new color and format. The user can assign task dependencies by right-clicking on the menu over the main task.

Kidasa Software's Milestones Etc. 5.0

Milestones reclassifies the subtasks as a main task when a main task is deleted. When the user moves the end point of an independent task, the start points for dependent tasks cascade to correspond with the new end point. Spell check is available.

Milestones has two presentation views: calendar and Gantt. Milestones can easily publish to the Web. The user can save a file as an HTML document and save schedules in a graphic format. Milestones can include numeric data as a part of its charting, but analysis of the data is not present. The tool has telephone and e-mail support. Milestones provides a question and answer page for installation and a top 10 support-question list. The major characteristics of Milestones Etc. 5.0 are as follows (Milestones Etc. 5.0, Kidasa Software Inc., Austin, Texas, 512-328-0167, www.kidasa.com):

- Hardware requirements include Windows 9x or NT, 32 Mb RAM, 40 Mb storage, and 133 MHz Pentium.
- Ease of installation is excellent.
- Data entry is fair.
 - ◆ Intuitive data entry format: Yes
 - ◆ Data hierarchy: No
 - ◆ Dependent tasks connected to independent ones: Yes
- Data editing is excellent.
 - ◆ Easy to go back and edit: Yes
 - ◆ Dependent data remain after removing data with a dependent task: Yes
 - ◆ Cascading time lines: Yes
 - ◆ Spell check: Yes
- Presentation is fair.
 - ◆ Charts easy to make: Limited
 - ◆ Gantt timeline: Yes
 - ◆ Calendar: Yes
 - ◆ PERT charting: No
 - ◆ Resource allocation: Yes
 - ◆ HTML file creating: Yes
 - ◆ Color publishing: Yes
 - ◆ Graphics creation from scheduling: Limited
 - ◆ JPEG: No
 - ◆ Bitmap: No
 - ◆ GIF: No
 - ◆ TIFF: No
- Advanced features are fair.
 - ◆ OLE available: Yes
 - ◆ Workload tracking: No
 - ◆ Cost and hour tracking: No
- Help information is good.
 - ◆ Telephone support: Yes
 - ◆ Online help: Yes
 - ◆ Web site: Yes
 - ◆ Hardcopy manual: Yes

AEC Software's FastTrack Schedule 6.0

FastTrack focuses on activity listing and does timeline editing. FastTrack lacks color options and table options.

The tool is easy to install. FastTrack puts an icon on the Start menu, on the desktop, and under the program file information. FastTrack also uses the Gantt format. The user can insert new lines to the task. The tool indents tasks to denote hierarchy. Dragging a task's timeline bar and connecting it to another can link subtasks. Editing the project data is not comfortable. As each task and subtask is added or edited, associated time lines disappear and the task reverts back to the default—the current day's date. The user must reenter all previously scheduled information.

The user can retype an activity's information and change start and end dates without changing data for other tasks. The time lines bars also can shift the time.

FastTrack reclassifies the subtasks as a main task when a main task is deleted. When the user moves the end point of an independent task, the start points for dependent tasks cascade to correspond with the new end point. Spell check is available. In presentation, FastTrack gives a Gantt view with total–day allocation below.

FastTrack can easily publish to the Web. The user can save a file as an HTML document and save schedules in a graphic format. FastTrack does workload tracking as a part of the standard schedule, but labor costs are not present, nor is there a way to allocate resources. The tool has telephone and e–mail support. FastTrack provides a tutorial, help topics, and an online FAQ list (the tool maintains an FAQ page). The major characteristics of FastTrack are as follows (FastTrack Schedule 6.0, AEC Software Inc., Sterling, Va., 703–450–1980, www.aecsoft.com; Jackson, 1999):

- Hardware requirements include Windows 9x or NT, 32 Mb RAM, 20 Mb storage, 133 MHz Pentium.
- Ease of installation is good.
- Data entry is excellent.
 - ◆ Intuitive data entry format: Yes
 - ◆ Data hierarchy: Yes
 - ◆ Dependent tasks connected to independent ones: Yes
- Data editing is excellent.
 - ◆ Easy to go back and edit: Yes
 - ◆ Dependent data remain after removing data with a dependent task: Yes
 - ◆ Cascading time lines: Yes
 - ◆ Spell check: Yes
- Presentation is good.
 - ◆ Charts easy to make: No
 - ◆ Gantt timeline: Yes
 - ◆ Calendar: Yes
 - ◆ PERT charting: No
 - ◆ Resource allocation: Yes
 - ◆ HTML file creating: Yes
 - ◆ Color publishing: Limited
 - ◆ Graphics creation from scheduling: Limited
 - ◆ JPEG: Yes
 - ◆ Bitmap: Yes

- ◆ GIF: Yes
- ◆ TIFF: Yes
- Advanced features are good.
 - ◆ OLE available: Yes
 - ◆ Workload tracking: Yes
 - ◆ Cost and hour tracking: No
- Help information is good.
 - ◆ Telephone support: Yes
 - ◆ Online help: Yes
 - ◆ Web site: Yes
 - ◆ Hardcopy manual: Yes

Digital Tools, Inc., AutoPlan

AutoPlan is a graphic project management system for open systems. With this tool the project team members exchange information across multiple platforms. The planners and engineers use the same platforms for scheduling projects that they use for design and engineering and rolling up project data for summary level reports. This allows the user to manage projects effectively by providing a powerful graphic user interface that supports multiple overlapping windows and a mouse. Time analysis, resource and cost analysis, work breakdown structure (WBS), and progress monitoring are easy.

The capabilities of AutoPlan are as follows (Digital Tools, Inc., 408–366–6920):

- Project management functional areas
 - ◆ Project scheduling
 - ◆ Resource management
 - ◆ Project tracking
 - ◆ Project reporting
- Project scheduling functionality
 - ◆ Critical path method (CPM)
 - ◆ Precedence analysis
 - ◆ Multiple project scheduling
 - ◆ Task priority
 - ◆ WBS modeling
- Resource management functionality
 - ◆ Conflict analysis
 - ◆ Profile definition
 - ◆ Resource leveling
 - ◆ Pool definition
- Project tracking functionality
 - ◆ Planned versus actual
 - ◆ Cost/schedule variance
 - ◆ Man–hour accounting
 - ◆ Contingency planning
 - ◆ Earned value (EV) analysis

Computer Associates International, CA–SuperProject for PC

- Project reporting functionality
 - ◆ Gantt chart
 - ◆ Resource histogram
 - ◆ Cost line graph
 - ◆ Network diagram

The platform requirements for AutoPlan are as follows:

- Sun Sparc; Sun OS: OpenLook, Motif
- HP 9000/400, 700; HP–UX: Motif; Motorola: MPC, UNIX, Motif
- IBM: RS6000; AIX; Intel: all, SCO, ODT/Motif
- DEC: 3100, 5000, ULTRIX, Motif; Data General: Motif
- Silicon Graphics: IRIS Indigo, IRIX 4.0, Motif

The minimum configuration for AutoPlan is as follows:

- Selected server with 16 Mb RAM
- Selected workstations with 8 Mb RAM

Computer Associates International, CA–SuperProject for PC

CA–SuperProject for PC is a project planning, tracking, and reporting tool that implements the Microsoft Windows graphic user interface. The tool incorporates multiple views for editing and display of project information. The tool provides dynamic data exchange (DDE) connectivity to external spreadsheets and other CA–SuperProject files. CA–SuperProject is packaged with CA–Realizer, a superset of the BASIC high–level programming language that enables the creation of external applications that can interface directly with CA–SuperProject. Standard report templates are provided with user customization capabilities included. CA–SuperProject generates Gantt, PERT, and resource histogram charts as well as cost line graph and spreadsheet EV data.

The capabilities of CA–SuperProject are as follows (Computer Associates International, Inc., 1–800–CALL–CAD):

- Project management functional areas
 - ◆ Project scheduling
 - ◆ Resource management
 - ◆ Project tracking
 - ◆ Project reporting
- Project scheduling functionality
 - ◆ CPM
 - ◆ Precedence analysis
 - ◆ Multiple project scheduling
 - ◆ Task priority
 - ◆ WBS modeling
- Resource management functionality
 - ◆ Conflict analysis
 - ◆ Profile definition

COSMIC, COMPASS

- ◆ Resource leveling
- ◆ Pool definition
- Project tracking functionality
 - ◆ Planned versus actual
 - ◆ Cost/schedule variance
 - ◆ Man-hour accounting
 - ◆ Contingency planning
 - ◆ EV analysis
- Project reporting functionality
 - ◆ Gantt chart
 - ◆ Resource histogram
 - ◆ Cost line graph
 - ◆ Network diagram

The platform requirement for CA-SuperProject is 80386 SX (16 MHz), MS-DOS 3.0, and Windows 3.0 or higher.

The minimum configuration for CA-SuperProject is 80386 SX (16 MHz), 4 Mb RAM, 10 Mb hard disk space, VGA monitor, Mouse or Track ball, MS DOS 3.0 or higher, and Windows 3.0 or higher.

COSMIC, COMPASS

COMPASS is suitable for a wide range of projects, including activity scheduling. The tool has the capabilities necessary for the user to plan creation in advance of execution time, plan revision before execution time, and plan revision at execution time in response to failure and delays. The tool can be used to manage activities that are subject to timing constraints, ordering constraints, and availability of resources. The tool can also be used to manage a variety of resources, including other tools, equipment, crews, electricity, and water. The resources may be used and then returned by an activity, consumed by an activity, or even produced or supplied by an activity.

The capabilities of COMPASS are as follows (COSMIC, 706-542-3265):

- Project management functional areas
 - ◆ Project scheduling
 - ◆ Project reporting
- Project scheduling functionality
 - ◆ Multiple project scheduling
- Resource management functionality
 - ◆ Conflict analysis
- Project tracking functionality
 - ◆ Planned versus actual
 - ◆ Man-hour accounting
 - ◆ Contingency planning
- Project reporting functionality

Quality Software Products Co., MasterPlan

- ◆ Gantt chart
- ◆ Resource histogram
- ◆ The platform requirement for COMPASS is Sun Sparc; Sun 3.
- ◆ The minimum configuration for COMPASS is as follows:
 - 4 Mb RAM; 2 Mb hard disk space
 - Sun Workstation with Sun OS 4.0
 - X–Windows release 11.3 or higher, VERDEX Ada compiler

Quality Software Products Co., MasterPlan

MasterPlan 1.1X is a graphically oriented project management system for UNIX–based systems running the X–Windows system. This tool can be used for planning projects of varying sizes. This provides project management features such as CPM, Gantt charts, PERT charts, resource allocation and leveling, and a variety of resource costing procedures.

The capabilities of MasterPlan are as follows (Quality Software Products Co., 1–800–288–6981):

- Project management functional areas
 - ◆ Project scheduling
 - ◆ Resource management
 - ◆ Project tracking
 - ◆ Project reporting
- Project scheduling functionality
 - ◆ CPM
 - ◆ Precedence analysis
 - ◆ Multiple project scheduling
 - ◆ WBS modeling
- Resource management functionality
 - ◆ Conflict analysis
 - ◆ Resource leveling
 - ◆ Pool definition
- Project tracking functionality
 - ◆ Planned versus actual
 - ◆ Cost/schedule variance
 - ◆ Contingency planning
 - ◆ EV analysis
- Project reporting functionality
 - ◆ Gantt chart
 - ◆ Resource histogram
 - ◆ Cost line graph
 - ◆ Network diagram

The platform requirements for MasterPlan are as follows:

- Sun Sparc; Sun OS: OpenLook, Motif
- HP 9000/400, 700; HP–UX: Motif; Motorola: MPC, UNIX, Motif

Cambridge Management Systems, Project Outlook

- IBM: RS6000; AIX; Intel: all, SCO, ODT/Motif
- DEC: 3100, 5000, ULTRIX, Motif; Data General: Motif
- Silicon Graphics: IRIS Indigo, IRIX 4.0, Motif

The minimum configuration of MasterPlan is 8 Mb RAM, 150 Mb hard disk space, and X–Windows 11.3 or higher.

Cambridge Management Systems, Project Outlook

Project Outlook is a project management scheduling design and control tool that uses the Microsoft Windows graphic user interface.

The capabilities of Project Outlook are as follows:

- Project management functional areas
 - ◆ Project scheduling
 - ◆ Project tracking
 - ◆ Project reporting
- Project scheduling functionality
 - ◆ CPM
 - ◆ WBS modeling
- Project tracking functionality
 - ◆ Planned versus actual
 - ◆ Project reporting functionality
 - ◆ Gantt chart

The platform requirement for Project Outlook is an IBM–compatible PC and Windows.

The minimum configuration for Project Outlook is Windows 3.0 standards.

Risk Radar

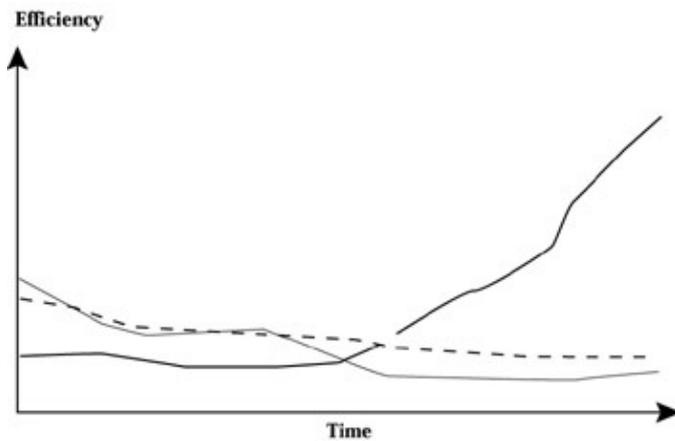
Risk Radar is a risk management application by the Software Program Management Network (SPMN) that identifies, organizes, prioritizes, tracks, and displays project risks. Project managers who use this tool can easily characterize, rank, and communicate project risk with the entire project team. Risk Radar keeps risks in view so that the manager can preempt risks before they become disasters. The control panel improves project management and avoids unwanted surprises by enabling software managers to monitor and control key drivers of project cost, schedule, and performance.

The Risk Radar tool helps the manager visualize and monitor project progress and status. It displays software development progress measurements that focus on time, money, staff, and defects. Two MS Excel 7.0 spreadsheets supply the data for the control panel: the first provides project data, and the second provides risk data. When these spreadsheets are regularly updated, the control panel supplies the information needed to keep the project on course, allowing every member of the project team to view the project performance. The control panel provides graphic depiction, clarity, concise information, and best practices. The control panel provides the project manager with the status of the following seven critical areas for monitoring purposes:

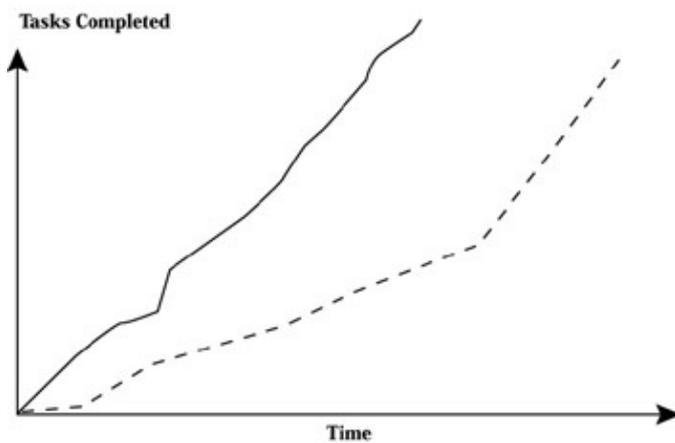
Cambridge Management Systems, Project Outlook

1. Progress
2. Productivity
3. Completion
4. Change
5. Staff
6. Quality
7. Risk

In addition, Risk Radar uses a standard Microsoft Access database, which contains the data and user interface in a single application file, which is accessible through File Manager in Windows 3.X or directly through MS Access (Versions 2.0 or higher) (Figure 8-1, A-C).

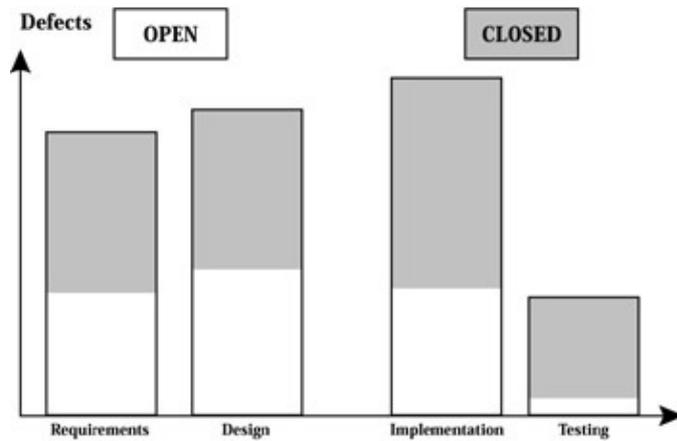


8-1 A: Project performance efficiency



8-1 B: Tasks completed

Commercial Items Checklist



8-1 C: Defects by phases

Commercial Items Checklist

- Do extensive market research.
- Select suitable CIs early in the project's development.
- Study product performance.
- Consider quality assurance of the CI (reliability, maintainability, and availability).
- Consider vendors' experience in addition to the product's cost.
- Evaluate market standards and practices.
- Establish flexibility in the operational requirement to allow consideration of a broader range of CIs.
- Document the results of market research.
- Scrutinize vendors' guarantees and warranties for the product.
- Analyze vendors' retention of technical data rights.
- Take advantage of vendors' training and support for the CI.
- Test the usefulness of CIs before making a commitment.

Section III: Using the Latest Technology

Chapter List

Chapter 9: Customer, Customer, and Customer

Chapter 10: Network Management

Chapter 11: Internet Applications

Chapter 12: Distributed Object Technology

Chapter 13: Distributed Objects

Chapter 14: Wireless Practical Case Study

I don't know the key to success, but the key to failure is trying to please everybody.

–Bill Cosby

Chapter 9: Customer, Customer, and Customer

The quality of IT system management is determined by customer satisfaction, not by the absence of system defects. Customer satisfaction is important for an organization to grow and stay in business. Customer satisfaction plays a vital role in building an organization's reputation and progress. The IT project manager wants to improve his or her organization's ability to build and maintain a loyal customer base with quality products and services. This chapter presents proven strategies that a manager should follow to build long-term relationships with customers that will create value for the customer and the organization.

Knowing the Customer

Customer is a generic word used in this chapter and is interchangeable with *user* and *stakeholder*. Knowing the customer is important for the success and prosperity of an organization. The IT project manager must know what customers want for the IT system and what they think of his or her organization and services (Figure 9–1). The manager should keep his or her customers and system developers well connected. Box 9–1 shows a list of features that will help the manager understand the customer.

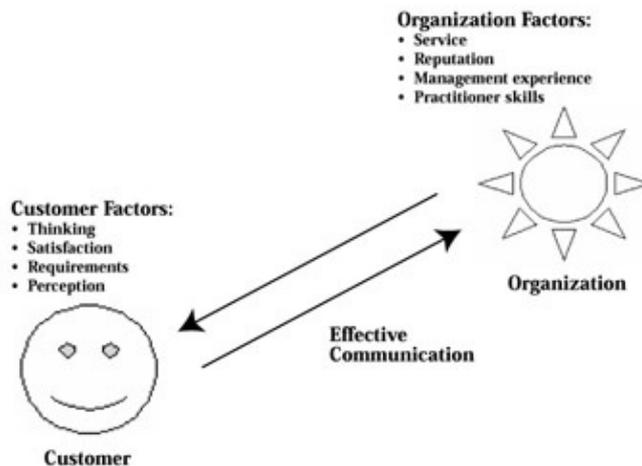


Figure 9–1: Knowing your customer's parameters

Box 9–1: Features to Help the Manager Understand the Customer

- Understand the customer's requirements.
 - Maintain open communication with the customer.
 - Include the customer in the organization profile.
 - Keep the customer in the loop for the project status.
 - Establish processes for delivering quality service.
 - Establish a point of contact who will respond to the customer's inquiries immediately.
-

Customer satisfaction and dissatisfaction results provide vital information for understanding customers. The project manager should listen to and learn about each customer on a continuous basis. The use of electronic commerce (e-commerce) is rapidly changing the industry and may affect the manager's listening and learning strategies and his or her definition of the customer. The manager's selection of listening and learning strategies depends on the organization's key business factors. Some frequently used strategies include focus groups with key customers, close integration with key customers, interviews with lost customers about their decisions, use of a customer-complaint process to understand key services, and survey feedback information, including use of the Internet.

Customer Relationship Management Process

The customer survey is important for quality customer service. One of the uses of a survey is to identify the level of customer satisfaction. Another use can be in areas that need improvement, such as the IT system environment, organization infrastructure, hardware and software, utilities operations, laboratory facilities, service orders, and individual tasking. A suggested customer survey rating scale is identified as follows:

5 = Exceptional

4 = Very good

3 = Good

2 = Poor

1 = Very poor

Customer comments are welcome to improve the relationship between the IT system developers and the customers. An organization cannot survive without customers.

The customer is satisfied if he or she receives the IT project information quickly and accurately. Customer satisfaction survey results give the manager vital information for understanding the customers. Such results provide meaningful information for the customer's views that provide repeat business and positive referrals for future business.

Customer Relationship Management Process

The customer relationship management process involves the project manager listening, learning, and implementing a performance excellence strategy. The average customer is an ordinary person who needs to buy what every successful retail store supplies. The manager should use common sense to win the customer. Respect begets respect. It is not difficult for the manager to establish a cordial relationship with a customer if he or she wants to stay in business.

The customer relationship management process uses data mining techniques, continuous data updates, and one-to-one communications programs. This process endures optimal handling of every customer contact. The success of a business depends on a good relationship with the customer.

With the rise of the Internet and e-commerce, customers are more empowered and more demanding than ever. They expect to be able to contact an organization the way they want and when they want. If the manager cannot provide customers with the information and services that they need, he or she will lose them as customers.

Customer Expectations

The project manager must meet customer expectations to be successful. A customer expects the IT system to be reliable, maintainable, and available, all of which help provide an index of system suitability. These measures are interdependent and must complement one another in the development process. Box 9-2 and Figure 9-2 list customer expectation factors.

Box 9-2: Customer Expectation Factors

Customer Relationship Management Process

- Study and planning
 - Quick response
 - E-mail facilities
 - Web page of the organization with information on the latest updates
 - Easy wireless access
 - Courteous reception that is the first contact with the organization
 - Help desk to answer customer questions
 - A toll-free telephone number for easy access
 - Cellular telephone capability
 - Voice over Internet protocol
 - System developers' vision by introducing latest technology
 - Text chat
 - Call back
 - Reduced risk in the system development
 - Guidance for developing the IT system more quickly, cheaper, cleaner, and with enhanced quality
-



Figure 9-2: Customer expectations

A system is *reliable* if it does what it is supposed to do at any given time. The reliable system provides information to decision makers quickly. The reliability of a system is important because an unreliable system can have drastic effects on the system's life cycle. The project manager can evaluate reliability by having a system operate for long periods of time without human intervention. The system should detect error conditions and then recover from them. This means that the system will perform the required function as stated in the conditions within the stated time. Reliability must be emphasized in all phases of the system engineering; it cannot be added after the system has been developed but must be introduced in the initial stages of the system development.

Maintainability means that the system modules, components, and units can be readily enhanced, adapted to new environments, and corrected. Operational *availability* provides insight into whether the system will be in place and working when it is needed for mission performance. *Statistical validity* means that the system's performance is consistent enough and differences are large enough to convince the manager that those outputs resulted from the inputs. *Design validity* means that the inputs were controlled enough for the manager to relate the output performance directly to the inputs of interest. *External validity* means that the results are applicable to real world operations. Table 9-1 shows the confidence level graphically.

Customer Acceptance Criteria

Table 9–1: Customer Confidence Level

Number of tests without any failures necessary to demonstrate specific lower confidence level reliabilities								
LCL	.90	.95	.98	RD .99	.995	.999	.9995	.9999
.90	22	45	114	230	460	2301	4604	23025
.95	29	59	149	298	598	2994	5990	29956
.99	44	90	228	458	919	4603	9208	46050
.995	51	104	263	527	1057	5296	10594	52981
.999	66	135	342	688	1379	6905	13813	69075

WHERE $N = \frac{\ln(1-LCL)}{\ln RD}$

Customer Acceptance Criteria

The project manager should clearly and precisely define customer acceptance criteria for the IT system. The customer should write his or her requirements, expectations, and the detailed process for accepting the system. Acceptance involves the proper planning and execution of tests that demonstrate that the system satisfies the customer's needs. The tests include functional, performance, and integration tests that will be developed during the system development. For their own satisfaction the customers perform the acceptance tests to see for themselves that the result achieved is correct. Figure 9–3 shows a customer acceptance scenario.

Customer acceptance scenario:

- Customer expects system guarantee and warranty.
- System is delivered on time and within budget.
- System is delivered with proper documentation.
- Organization has available technical support, if needed.
- Technical help is available on Internet and e-mail messages.
- Organizations have established help desk.
- Response time is quicker to seek help.
- User training has been provided.
- System is providing correct information and faster for decision makers.
- No copyright or proprietorship by system developers exists.
- System is easy to use.

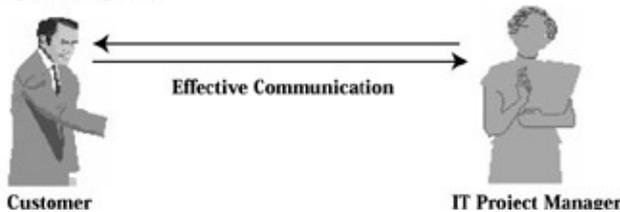


Figure 9–3: Customer acceptance scenario

The following are some of the tools used for acceptance tests:

- Coverage analyzer
- Timing analyzer
- Standards checker

The customer establishes that the system documents will be online and in electronic formats beside the hard copies. The customer can also establish constraints and conditions for the system so that it will meet all of his or her expectations.

Case Study 9–1

Customer confidence levels are important. For example, a contractor makes a change after a failure, tests it once, and says it is fixed. The customer should ask for a confidence level because the sample size is one and the confidence that the system has achieved a certain reliability requirement is probably very low. Using binomial distribution shows that a reliability requirement of 0.99 is achieved with a 90% lower confidence level if 230 trials are completed without a failure.

Customer Equation

The customer equation consists of a full range of human factor engineering (HFE), manpower, personnel, training, health–hazard assessment, and system safety. These factors improve customer performance and total system performance throughout the system development and acquisition processes. The customer equation contains the following:

- Manpower and personnel integration
- HFE
- Human system integration (HSI)

Manpower and Personnel Integration

Manpower and personnel integration (MANPRINT) is an umbrella concept encompassing HFE, manpower, personnel, training, health–hazard assessment, and system safety. The result of using the MANPRINT concept is that the system development and maintenance life cycle operates in the most cost–effective and safest manner consistent with the manpower structure, personnel aptitude and skill, and available training resources. The MANPRINT requirements are based on a team concept. The MANPRINT team's methodology is to compare and balance the following factors for the customer's satisfaction:

- Software engineering cost
- Hardware engineering cost
- Personnel quality constraints
- Operator/user ability
- Training requirements
- Overall system performance
- System reliability, maintainability, and availability
- System security

The goal of the MANPRINT program is to deliver an IT system that meets the customer's technical and human engineering requirements. The project manager establishes the following:

- Tasks and milestones for planning, coordinating, and integrating the MANPRINT elements
- The system hardware and software engineering team responsible for the design and development before various phases
- An analysis of evaluation data to determine if the MANPRINT requirements are achieved

The MANPRINT program is the key to providing an operator–trainable engineering product. The manager compiles all minutes regarding the status progress report for the MANPRINT program and submits it to the customer at all formal system reviews.

Human Factor Engineering

HFE ensures that the designed system meets the customer's requirements and will be operated and maintained within the capabilities and skill levels of the expected users. HFE takes into account the following factors:

- Human characteristics
- Skill capabilities
- Performance
- Safety factors
- Training
- Staffing implications

The manager develops the HFE data during the system concept exploration phase to do the following:

- Determine probable human performance, health and safety requirements, and projected personnel requirements.
- Develop planning of personnel support and training programs.
- Support organizational and operational concepts and provide all requisite HFE input to the management plan.

The manager progressively applies the HFE management plan to the system engineering and maintenance phases. Development of HFE inputs requires a thorough engineering analysis of system requirements. Personnel and training planning describes actions, decisions, processes, and procedures necessary to staff the organizations that employ or support the system to customer satisfaction.

Training development plans address all training support for a specific system, including personnel requirements, training device requirements, and the influence of the system on other aspects of training. The project manager plans to provide resources to accomplish HFE, personnel, and training.

The human equation helps the manager boost the morale of those who are involved in the development, maintenance, and operation of the system engineering.

Human System Integration

HSI involves all of the other terms and is not just HFE or system integration as is often perceived. HSI is concerned with all aspects of personnel, the environments in which they must work, the equipment with which they work, and the interfaces with each.

Evaluation of the personnel aspects of a system during a test requires skilled test design processes and experienced human factor personnel. The manager asks the following questions during the evaluation:

- Why was the operation performed poorly? Were the personnel not skilled enough, smart enough, or strong enough? Was the system not designed to the skill level of the operators in the force?
- Was it a personnel problem or a training problem?
- Were there not enough people to do the job (force structure/manpower issue)?

A correlation exists between the number of people required, basic skill and ability levels, and the amount of training that it takes to get those people to the necessary level to perform the job successfully.

HSI influences all areas from basic system technical performance (unless the system can perform without the

System Safety

people!) through operational effectiveness and suitability to the appropriate tactics and techniques given the user/maintainer population. Poor HSI design can have a negative influence on every aspect of system performance.

System Safety

Those responsible for system safety introduce the necessary safety measures for systems to ensure that the system executes within a system context without resulting in an unacceptable risk. The objective is to equip system–engineering practitioners with the extra knowledge and skills necessary to participate in safety–critical system development.

Safety is freedom from hazardous conditions that can cause fatal injury, damage, and loss of equipment and property. The following guidelines can help the user achieve safety:

- Design to prevent or minimize the occurrence of hazards.
 - ◆ Monitoring
 - ◆ Automatic control
 - ◆ Lockouts
 - ◆ Locking
 - ◆ Interlocks
- Design to control hazards, if they occur, using automatic safety devices.
 - ◆ Detection
 - ◆ Fail–safe design
 - ◆ Damage control
 - ◆ Containment
 - ◆ Isolation
- Provide warning devices, procedures, and training to help personnel react to hazards.

The techniques used to establish system safety standards depend on the direction of the management goals, customer's requirements, and importance of the system–critical mission. Management can set up a group to deal with safety concerns. The role of the system safety group is to do the following:

- Establish the system safety plan.
- Become involved with all of the safety measures.
- Analyze the safety measures.
- Attend all system engineering reviews to introduce safety measures.
- Attend configuration management (CM) board meetings to establish steps required to take system engineering safety measures for CM.
- Establish audit trails for all identified hazards.
- Monitor audit trails for safety measures.
- Plan and produce documents about how to take necessary steps in case of safety hazards.
- Design and issue necessary instructions to avoid hazards during system engineering development and maintenance.
- Produce necessary documents for safety precautions.
- Set goals to ensure that the system is safe and can operate in the presence of faults.
- Show that no single fault can cause a hazard in the system's safety and that hazards from sequences of faults are sufficiently unlikely.
- Identify the critical safety areas of the system for future monitoring.

User–Machine Interfaces

- Analyze system engineering hazards for detection, elimination, control, and limitation of damage in case of an accident; methods in which the system can fail safely; and the extent to which failure is tolerable.
- Establish measures to prevent hazards:
 - ◆ Minimize complexity
 - ◆ Separate safety–critical functions and data
 - ◆ Limit actions of the system
 - ◆ Minimize interfaces
 - ◆ Incorporate firewalls
 - ◆ Limit authority
 - ◆ Limit access
 - ◆ Minimize the duration of hazardous states
 - ◆ Control flow limitation
 - ◆ Control sequences such as concurrency, synchronization, and hand–shaking
 - ◆ Protect against credible hardware and software failures.
- Adopt a structured approach.

User–Machine Interfaces

The user–machine interface is important for IT system engineering development and maintenance. Human factors and system requirements are to be interfaced in a friendly atmosphere so that the system product is user friendly. This approach builds confidence in those who have to work with the system.

System Security

System security is important for all systems. A good security schema for information data may employ many security methods customized to an installations particular needs. The manager should take precautions to prevent hackers from penetrating the system and destroying the data. The following are some of these security methods:

- **Data encryption.** Data encryption consists of software and hardware techniques that safeguard data. The encryption is imposed upon the data by mathematic operations and a certain set of operands called the key. The key is mathematically represented as: $\text{Data} \oplus \text{Key} = \text{Encryption data}$
- **Cryptography** is introduced as a means of transforming data information into a form in which its meaning is obscured. The mathematic operation can be made as complicated as time permits to obscure the data information. Data encryption keeps data information secret and authentic. The same key and the reverse mathematic operation is required at the receiving end to decode the data.
- **Data compression.** Data compression reduces the number of bits necessary to represent a character for line–speed purposes. Encryption and data compression both may involve bit manipulation.
- **Audit trails.** Audit trails provide monitoring and tracking of errors and unsuccessful attempts to violate access privileges. This creates a comprehensive record of all network and system activities. This also provides statistical information of who logged on when and where and who is accessing what and for how long.
- **Passwords.** Passwords are a common, inexpensive technique for restricting access to data. They provide a means of authenticating the authority of a person to access certain types of data.
- **Security levels.** The use of security levels helps an organization classify its data in categories, such as top secret, secret, confidential, restricted, or unclassified, and to determine each categorys backup needs.

User–Machine Interfaces

- **Access levels.** Access levels can be assigned to users for a variety of access privileges. Some can have access privileges to update the data dictionary and database files, and others privileges can be more restricted. Other examples of access levels are read–only, write–only, remote access, specific file or directory accesses, ability to upload or download data from a mainframe or network database, and access to printer capabilities.
- **Lockout.** Lockout uses a password to get at a specific resource. All users of that resource must know the password.
- **Callback.** Callback devices prevent unauthorized access to the communications channel of a computer. Some devices require a caller to provide an identification number and hang up. After verifying the users access rights, the device calls the user back. This technique can also be combined with the use of passwords.
- **Verification and authentication.** Verification and authentication is a method of verifying that data are not altered between the time they are sent and received.
- **Biometrics.** Biometrics are high–technology devices used to measure some physical characteristics of the individual seeking access, such as fingerprints or retinal patterns. Biometrics measures are often combined with other measures, such as personal identification numbers (PINs), to determine authorization rights.
- **Smart cards.** A smart card is a card with magnetic strips or embedded chips that contain access information. After inserting the card into a reading device, the user types in a PIN to gain access.
- **Management of Customer Training.** Training of customers and users is of prime importance in achieving maximum system productivity. The users training includes the means of efficiently operating the system, the things that can go wrong, and when appropriate, how to recover the intelligence information. The training also covers precautions to be taken to protect the data. The users training effort commences at the beginning of the systems concept and continues throughout the systems development life cycle until all training products have been delivered. The training material consists of the following:
 - **Task analyses.** Task analyses reflect operator actions influenced by system changes. The task analyses are prerequisites for all other training products. Material gathered during the system concept phase and material contained in the Engineering Change Proposal documentation are the source material on which to base changes to the task analyses. The task analyses documentation is delivered at the time of system delivery.
 - **Technical manuals.** Technical manuals reflect the appropriate system changes. The Engineering Change Proposal and other documents are the basis for changing and updating each technical manual. The technical manual is delivered either as a total revision or on a page–by–page basis. Those changes are developed and validated as shown in the appropriate version development schedule. Camera–ready copies of technical manuals are delivered concurrently with delivery of the system.
 - **Instructor and key personnel training.** Instructor and key personnel training is developed to reflect operational changes occurring on each version resulting from implementation of version requirements. The training is conducted in the users facility. The course material is delivered to the user, and videotapes are produced as needed.
 - **Operators notes.** Operators notes are developed to instruct operators in how to handle the system. The operators notes contain commands and instructions, including the following:
 - ◆ Start
 - ◆ Stop
 - ◆ Initialize
 - ◆ Print reports
 - ◆ Follow diagnostic guidelines
 - ◆ Perform recovery techniques
 - ◆ Perform back–up procedures
 - ◆ Switch actions

Case Study 9–2

- ◆ Fixing minor problems
- ◆ Reboot

Training, personnel, and manpower are not independent. A test generally indicates whether a task was performed to a standard, but determining why it was not is more difficult. The training program can be focused on the wrong population, it can be too short for the given population, or it can be a poor training program.

In many instances, the training program is late in being developed for new systems because the new system is late in its own development cycle. The training developers have not had their hands on the system to see how it actually works and how the target user population will use the system in practice. This new system will be compared with an existing system that has been in use for several years and has been optimized by experience. Work-around has been developed and documents have been modified to fit the capabilities of the system. Customers prefer online training as well.

Case Study 9–2

Mr. Jacobson focuses on developing the USPSs information platform, systems integration, data warehousing, and e-commerce initiatives. The technology office also continues producing Point of Service One retail terminals.

Jacobson said that his plans include making it possible for postal customers to track mail via several USPS websites. The service will also use mail tracking as an internal management tool.

Customer Management Checklist

- Have a clear business plan for interaction with customers.
- Assign responsibility to key officials to work with customers for their satisfaction.
- Show courtesy, dignity, and respect to customers.
- Educate all employees, especially receptionists, about the organization. Receptionists are the first people in an organization to meet customers (the first impression is the last impression).
- Emphasize quality of class-one service.
- Provide proper training to the work force so they will be courteous toward customers.
- Establish a long-term relationship with the customer.
- Move aggressively on IT system development by forming a team with the customer.
- Provide information to the customer when he or she needs it.
- Establish a knowledge management concept that provides systematic attention to and sharing of knowledge with developers and customers. The key to knowledge management is the simple idea that many heads are better than one.
- In case a customer is lost, reevaluate business practices and make necessary changes for the future.
- Develop a partnership between the customer and the IT system developers.

Chapter 10: Network Management

Current IT project systems have requirements for networking so that the data can be accessed electronically from a distance. The IT project manager should understand the basics of network management for effective IT project planning.

Telecommunications Basics

Telecommunications is defined as the electronic transmission of information. An example of telecommunications is the combination of a computer with a link for two-way communications with other computers. Telecommunications is the art and science of communicating at a distance, especially by means of electromagnetic impulses, as in radio, radar, telegraphy, and telephone. The transmission of information can be voice, data, and pictures. Improvements in teleconferencing, voice and electronic mail (e-mail), electronic bulletin boards, satellite links, fax machines, fiber optics, and information banks are being developed in telecommunications technology. Microwave towers are visible for transmitting voice and data. Fiber optics cables are replacing copper cables because of increased transmission speed and capacity and reduced size. Multimedia and compound documents that combine written text, voice, and computer communications are being developed.

A computer sends and receives data and information in the form of bits. A bit is an electronic pulse (1) or the absence of a pulse (0). Within a computer, data are stored and processed in a parallel form 16, 32, 64, 128, or 256 bits at a time and in a serial form 1 bit at a time. The computer's serial port (RS 232) converts parallel data into a serial form for transmission or converts serial data back into a parallel form.

Modems

A modem is an electronic device that carries and translates from binary to analog forms of data. The modem converts data from a binary on-off digital state that is used in the computer into an analog equivalent that can be telecommunicated. The name *modem* is derived from the conversion of digital data into analog data through a process known as *MODulation* and the reverse process known as *DEModulation*. The speed at which modems can send and receive information is measured by the baud rate. The baud rate is equivalent to the number of bits per second (bps) that the modem can transmit.

When a modem is used to connect a personal computer (PC) to a telecommunications network, the communication can be simplex, half-duplex, or full-duplex. When the communication is in only one direction, the mode is *simplex*. Communication between computers is in the *half-duplex* mode if only one computer can send information at any one time. The communication is in the *full-duplex* mode if both computers can send information at the same time. The *protocol* is the set of rules that two computers follow in sending and receiving information.

The telecommunications industry is evolving quickly. With the increasing use of fiber optics, wireless communications, and satellites as communication media, technology is evolving by replacing analog transmission with digital voice transmission. The transition to a digital standard, called an *integrated services digital network* (ISDN), makes possible the transmission of many types of signals (e.g., voice, video, and data). IT managers are faced with the herculean task of managing a mix of local area networks (LANs), Intranets, and Extranets. IT managers must keep up with the rush of bandwidth being used by e-mail, videoconferencing, and multimedia applets running across the Internet (Hammond, 1999).

IT Networks

IT networks use telecommunications to link two or more computers and allow them to share data, spread out the processing chores, or serve as huge repositories of information that is valuable to users. IT networks can be a LAN or a wide area network (WAN).

LAN

LAN is a system of computer hardware and software that links computers, printers, and other peripheral equipment into a network at a single location. LAN is suitable for transmission of data between offices in a building or between a range of buildings situated near one another (e.g., a university campus or an office building).

E-mail on a network is sending and receiving of messages, pictures, and documents to one or more persons. Linking PCs together with LAN can be difficult, especially if the PCs are not the same type or they are using different operating systems. The file server contains data and information files that other users can access and reuse.

LAN is configured as star, bus, or ring networks. In the star network, a host computer has multiple slave computers connected to it. Since the host controls all data communications, any communications between any two computers must be routed through the host computer. If the host fails, then the entire system goes down. Multiple PCs hooked to a file server form a star LAN.

A bus network has computers that tie into a main cable or bus with no one central computer. The failure of any one computer does not affect the overall performance of the network. To send a message to another computer requires only that the software be able to signal the correct computer to receive the message. An example of a bus network is the popular Ethernet system.

In a ring network, all computers are treated the same and intervening computers must process communications between two computers. To avoid having the system go down when a single PC fails, ring LANs often have ways to drop the single PC without interfering with the operation of the LAN.

WAN

WAN can range from a few city blocks to almost global. WAN consists of academic and research networks (private networks designed to provide communications between an organizations host computer and the employees and customers terminals) and value-added networks. All WAN computers use packet switching for telecommunications. In packet switching, the terminals are linked to the host computer through interface computers. The host computer breaks up long messages into data units called *packets*, which are then given to the interface computers to transmit through the network. The terminal at the destination receives the packets and reassembles them into a copy of the original message.

Academic and research networks interact between universities and other research institutions, like the Advanced Research Projects Agency Network (ARPANet). Commercial e-mail systems allow subscribers to send and receive electronic messages via the Internet.

SuperNet is the network being designed by a team of U.S. government, academic, and industry researchers to achieve a high-speed nationwide network that offers a glimpse of the Internets future. The network will provide end-to-end transmission speeds of 2.5 million bps more than 1000 times faster than the current Internet. The program is responsible for commercialization of high-speed optical switches currently being

Internet Technology

deployed by a long-distance carrier. The next phase of the research program will focus on protocols and network management tools that will guarantee end-to-end quality of service and security for high bandwidth applications.

The SuperNet backbone will connect 100 users via six regional research networks that range in speed from 2.5 million bps to 20 million bps. Researchers will be able to reserve huge amounts of bandwidth for demanding real-time applications, such as telemedicine, distance learning, and videoconferencing. The network management algorithms and control strategies that result from this effort will make it easier and more cost-effective for corporate customers to run high-bandwidth applications (Marsan, 1999).

Internet Technology

Internet technology consists of connected computers that allow information to be sent and received around the world. The Internet offers access to the World Wide Web (WWW). Wireline leverage technologies, such as hypertext transfer protocol (HTTP) for display of active server pages (ASPs) with dynamic content in the form of hypertext markup language (HTML) or extensible markup language (XML), optimize user experience.

Wireless Technology

Wireless technology means accessing information anytime, anywhere. Accessing real-time organization data and e-mail and facilitating enhanced communications among practitioners are valuable advantages of wireless technology. In the United States, wireless technology has evolved from first generation to second generation.

The first generation is analog cellular, which consists of circuit-switched data (CSD), including a voice channel dedicated to data (just like a landline modem or a fax machine) and the ability to connect a phone via a cable to a notebook computer or personal digital assistant (PDA). Because of a dedicated connection, message length is not limited. An advantage of CSD is that dedicated connection time billing makes sense for larger files if speed is decent. However, this is a potentially costly way to use the Internet because CSD speed is in the range of approximately 4800 bps to 9600 bps and users are normally paying on a per-minute basis. Thus for short messages CSD can be inefficient.

The second generation of wireless technology consists of time division multiple access (TDMA) and global system for mobile (GSM) communication. An analog line dedicates a channel to a single user, whereas digital subscribers take turns using a channel. The TDMA speed range is from 9600 bps to 64,000 bps. Therefore the number of users may be increased by three compared with analog because three conversations share the same frequency. A benefit of TDMA is that battery life is extended because the phone transmits one third of the time. TDMA technology is the foundation for digital but is not a global standard because it is not used in Europe at this time.

GSM is similar to TDMA, with worldwide roaming and data rates of 9600 bps. It is the standard for most cellular systems on the personal communication service (PCS) band. Currently, over 130 countries that use GSM use GSM1900 (on different frequencies), code division multiple access (CDMA), and TDMA.

Challenges to the Project Manager

Although the value of wireless technology is limited because of issues such as slow connection speeds and user mobility, it can be a tool to complement a wireline application. The wireless web uses wireless application protocol (WAP), handheld device markup language (HDML), wireless markup language (WML),

Wireless Data Providers

HTML, and other emerging standards and tools. Disadvantages are high cellular phone rates, limited screen space, and virtually no graphic capability. The challenge for the project manager is to work with both information and technical architects to build an application and a user interface. This will deliver content at a low cost to the mobile user while continuing to provide rich content and full functionality without budget cost overruns (www.wapforum.org).

As a project manager, a web site may not need to employ an open architecture; however, it may be advantageous to consider an architecture that costs the client more up front but is able to handle a wireless component, especially as applied functionality improves. Verification of transmission, or letting the sender know that the receiver got the message, is important. Equally important is authentication, or verification of the identity of the sender.

Wireless Data Providers

Paging

The paging network delivers a broad array of coverage for a low volume of data at a relatively low cost. Paging comes in three types: one-way, two-way, and 1.5-way paging. One-way paging is the least expensive of the paging alternatives, offers the best coverage, and is one of the easiest solutions to deploy. However, it is only viable for sending information to users; it provides no way of verifying whether a message was delivered. Most of the same limitations apply to 1.5-way paging, with the notable exception of its being able to acknowledge the receipt of a message.

All devices using the two-way network are automatically switched between basic and full coverage, and the user is notified when the coverage area changes. Among the paging solutions, two-way presents the most compelling options. Truly interactive applications can be developed to facilitate effective two-way interactions. The carrier that offers two-way paging is somewhat limited as compared with one-way and 1.5-way.

Packet

The two major caveats for paging are speed and cost. Most paging networks perform at less than 2.4 kilobytes (kb) per second, and this is only viable for a small amount of information. Although paging networks are useful for sending small amounts of information, they are not reliable for mission-critical applications and were not designed for data transfer. To address these issues, two packet data networks were developed specifically for mission-critical wireless data needs. Because of the tremendous cost of deploying these networks and the limited base of customers demanding this type of service, packet data networks are expensive. The advantages of packet data over paging are the intrinsic error detection and correction capabilities for which these networks were designed to handle.

Cellular

Although cellular data networks were built for voice data transfer, the migration toward cellular data networks is progressing at a staggering pace. Cellular data networks offer several advantages over paging and packet data networks. The primary advantage is that cellular networks can effectively carry both voice and data on a single device. However, wireless data access using cellular data networks is expensive.

Satellite

Few wireless applications justify the cost and limitations inherent in satellite systems. The only advantage of using satellite networks for wireless data communications is for remote areas that cannot be covered by any other network (Hayden, 2000).

Wireless Devices and Platforms

Application interfaces are developed for multiple devices and platforms. For example, a PDA has a larger viewing screen than a mobile phone. Some of the wireless devices and platforms are discussed as follows:

- Wireless application protocol (WAP)enabled digital phones
- PDAs and handheld computers
- Windows CE
- Two-way pagers
- Global positioning system

WAP-Enabled Digital Phones

Wireless information services may be deployed on WAP-enabled telephones. WAP is a new standard but is becoming a widely accepted protocol. Developers prefer a single standard because they can write to a single open standard and carriers do not have to support multiple communication protocols, manufacturers, and users who do not have to make a corporate or personal decision about devices with one protocol or another. WAP is an open standard that allows wireless devices to communicate independently of vendor or wireless networks and thus can more easily access information and Internet services.

WAP is a network-independent standard that will operate on GSM, CDMA, and TDMA. It offers secure access to the Internet and is optimized for low bandwidth, high latency, and stability. Although the limited display size may not allow all of the content available on other hardware platforms, WAP-enabled telephones provide a broad selection of devices and networks and are already becoming prevalent among a wide user base. However, they are currently not supported by many devices.

PDAs and Handheld Computers

The Palm Vx runs the standard Palm operating system and may be used for other Palm-certified applications, such as address books and calendars. The Novatel Minstrel V modem uses any of the cellular digital packet data (CDPD) wireless networks in the United States. The Palm and the modem contain rechargeable batteries.

The Windows CE A Win CE HPC device coupled with a Sierra Wireless AirCard-300 will be used as one of the hardware platforms. The HP660-compatible device must be on the list of approved devices for the AirCard-300. The AirCard-300 is a CDPD PCMCIA modem containing external rechargeable batteries, but it also is able to draw power internally from certain Win CE HPC devices.

Two-Way Pagers: RIM 850 and 950 Pagers

The Research-In-Motion (RIM) 950 interactive pager can use the BellSouth Wireless Data (Mobitex) network and the Blackberry messaging service. RIM also makes a model 850 version of the pager, which operates on the ARDIS network. The 850 has a different and broader coverage area in the United States. Future devices from RIM are anticipated and may become ideal candidate platforms for wireless information services. As paging support for GPRS wireless networks becomes available, these devices will become a significant option for international usage as well (Motorola, RIM). The strengths of the RIM pager are the

Wireless Devices and Platforms

small size and light weight of the device, large penetration, and two-way capabilities such as e-mail, Internet access, and messaging. The weaknesses of the RIM pagers are their small work and display area, limited usage, and difficulty of data entry.

Global Positioning System

The global positioning system (GPS) determines the code phases (pseudoranges) for the various GPS satellites. GPS determines the time of applicability for the pseudoranges and demodulates the satellite navigation message. Users compute the position of the receiving antenna using the pseudoranges, timing, and navigation message data. The satellite navigation message and its inherent synchronization bits are extracted from the GPS signal after it has been acquired and tracked. To serve a larger area, the GPS reference receiver is replaced by feeds from a wide area reference network (WARN) of distributed receivers.

The WARN server retrieves and stores data from the GPS reference receivers and provides aiding data to mobile units. The server also performs navigation solutions (calculates longitude, latitude, and altitude) after receiving pseudorange measurements from the handset.

Enhanced GPS (EGPS) improves on conventional GPS performance by offering the following (Krasner, 1999):

- Allows improved sensitivity and acquisition times and enhances system accuracy by using a snapshot and the processing power of digital signal processing (DSP)
- Shares processing functions between the DSP built into most wireless handsets and a network server operating within a carriers network
- Uses information available from the wireless network itself
- Processes only a snapshot of data in software rather than processing it continuously in hardware

Uses of Wireless Technology

Adults in the United States use wireless technology for the following: 67% access e-mail remotely, 67% use GPS to get directions, 63% browse the Internet, 59% take digital pictures and send them to family members and friends, and 57% conduct a videoconference (*USA Today*, January 2000).

Future of Wireless Technology

The wireless market for web-enabled devices (e.g., mobile phones, PDAs, and two-way pagers) is growing at a much faster rate than the Internet. The research firm Yankee Group predicts that the number of worldwide wireless subscribers with Internet access will reach 15 million by the end of 2000. Motorola has shipped close to 5 million wireless Internet-enabled phones. Sprint has more than 7 million wireless customers, including about 2.5 million with Internet-enabled phones. AT&T has 12.2 million U.S. wireless users. Nokia began selling its Internet-enabled phones in the United States in the Fall of 2000. By 2003, more than 500 million data-enabled mobile phones, more than 20 million PDAs, and more than 100 million pagers will be in the marketplace, leading to a wireless base of 1 billion. In 1998 the PC market install base consisted of 298 million, and the expected growth by 2003 is an install base of only 550 million.

The future of wireless technology is to move to packet-switched connections and combine high-bandwidth and high-speed mobile access with wireless Internet services. This enables much richer applications, including videoconferencing, Internet and Intranet access, instant messaging, and interactive application sharing. This will initially be able to leverage existing spectra but ultimately will need new spectra to achieve necessary speeds.

Cellular Communications

The future of wireless technology will focus on increasing data flow and voice capacity with an eye on eventually replacing the wired phone service and continue challenges that include security issues, seamless data transmission, and an unprecedented demand for communications services.

Cellular Communications

Cellular communications relies on specific radio frequencies to carry information. Three frequency bands 800 MHz, 1900 MHz, and 2400 MHz have been designed by the U.S. Federal Communications Commission (FCC) as available for cellular services.

When a user activates a cellular phone, the signal travels to a tower equipped with a receiving antenna. The area covered by the antenna is called a *cell*, and cells vary in size, depending on terrain and capacity demands. Service providers place the equipment in locations so that coverage areas overlap.

As a user moves from one towers range to another, a process known as a *hand-off* takes place so there are no transmission breaks. The antenna works with a special-purpose computer, known as a *mobile telephone switching office*, that monitors all call activity. The computer also has an output line that feeds into a wired telephone line, connecting a caller to the public telephone network.

Cellular equipment suppliers have begun working with standard bodies to boost transmission speeds. The wideband CDMA standard, which is used to distinguish data transmission from voice transmission on cellular networks, should enable manufacturers to increase the top speeds of cellular signals from 14.4 to 64 kb/sec on uplinks and to 384 kb/sec on downlinks.

Virtual Private Networking

Virtual private networking (VPN) technology uses tunneling, encryption, authentication, and access-control technology to transmit data securely over the Internet or another public backbone, such as a service providers Internet protocol network or a providers backbone. VPN is quickly becoming a secure, cost-effective communication solution for a wide array of applications. VPN is replacing modems and leased lines as the preferred method for remote access.

The three types of VPN are remote access, Intranet, and Extranet. A remote-access VPN connects telecommuters or other mobile users to an organizations internal network; an Intranet VPN connects fixed locations, such as branch offices, to the internal network; and an Extranet VPN gives an organizations business partners, such as contractors, limited access to internal networks.

The classic remote-access use of VPN technology illustrates how the technology works. A remote user dials into a service provider and establishes a link to an organizations headquarters over the Internet or the providers network. The user then authenticates himself or herself for authorization to gain access to the organizations internal servers.

VPN technology is based on tunneling technology, which is used to transfer data between two similar networks over an intermediate network. One type of data packet encloses or encapsulates another data packet to shroud it from potential electronic eavesdropping, thus creating a private tunnel over a public backbone. The packets are encrypted so that the data are protected from an authorized user who may try to capture them during transmission (*Federal Computer Week*, p 17, July 19, 1999).

Chapter 11: Internet Applications

In the short existence of the Internet, a wireless aspect has spun off to provide another medium with which to surf the World Wide Web (WWW). Although still limited in its capability, one can no longer discuss the Internet without differentiating between wireline and wireless, which is a relatively new medium. This chapter presents electronic commerce (e-commerce) and mobile commerce (m-commerce) in relation to the Internet.

Emerging Wireless Technologies Standard

Wireless application protocol (WAP) is a communication protocol between a mobile terminal and an IT system (e.g., Internet, Intranet, corporate IT network), providing mobile access to services in a general environment for wireless applications and content. WAP uses wireless markup language (WML), a nonproprietary wireless language as shown in Figure 11-1. WAP is emerging as a standard that is independent of any single carrier or device. The objective of WAP is to stimulate a new generation of wireless services and products. The following are components of WAP:

- **Microbrowser on client device.** The microbrowser makes a request in WML. The request is passed to the WAP gateway (retrieves information from the Internet server) either in standard hypertext markup language (HTML) or WML. If the content is in HTML, a filter in the WAP gateway may try to translate it into WML. Requested information is then sent from the WAP gateway to the WAP client using available and appropriate mobile network bearer services. WAP corresponds to HTML (handheld device markup language [HDML] is proprietary to Phone.com).
- **WAP gateway/server.** The WAP gateway/server translates requests from WAP protocol to a WWW protocol stack.
- **HTTP and transmission control protocol/Internet protocol (TCP/IP).** Content encoders translate Web content into a reusable format.
- **WAP software tool kits**
- **WAP simulators**

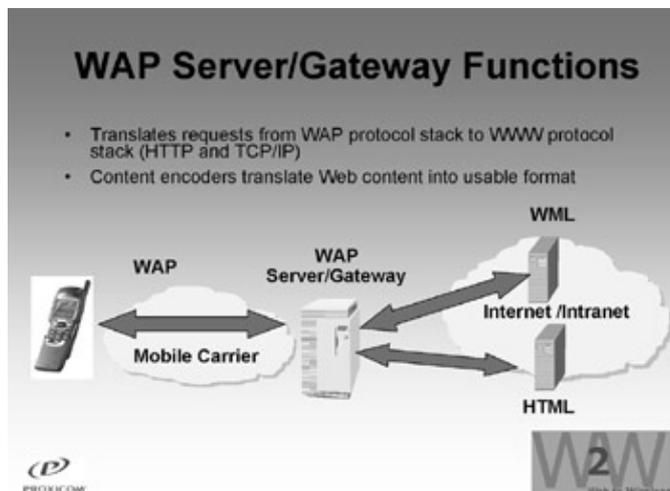


Figure 11-1: WAP server

Wireless Applications

Wireless technology is emerging, but because of issues such as slow connection speeds and limited user mobility, the value is limited. However, a wireless application can still be a valuable tool that complements a wireline application. For wireless applications, each transaction triggers a phone charge; thus a charge accrues each time an individual transmits or receives data. Given the limitation of keyboards with wireless devices, a highly interactive application can be expensive and difficult to use. However, opportunities exist in customer personalization at multiple levels by profiling (e.g., e-mail), personalized business data, personalized shopping, reservations, and ticket purchases. Another advantage of wireless applications is the sending of personalized messages, such as an airlines flight status (e.g., gate change or change of arrival and departure times), to customers on a mobile device. The goal is to require minimum data entry.

Current applications in the areas of wireless banking, stock trading, order entry, and shopping are focused at specific target audiences, are available through a few vendors, and are increasing. Other markets that are applying wireless technologies are mobile enterprises, field services, transportation, financial services, banking, retail, utilities, telemetry, health care, insurance, travel, location-based services, ticketing, auctions, entertainment, and real estate.

Wireless mobile enterprises include a mobile sales force; sales force automation; account, contact, and client database information; access to product inventories; real-time ordering of products and services; checking on previously ordered products; transmission of expense forms; and access to client databases.

Extending the office desktop provides access to corporate databases and a constantly connected work force. Mobile workers need access to the corporate database, e-mail, address and phone books, calendars and GroupWare, and mobile devices in the corporation.

E-Commerce and M-Commerce

E-commerce and m-commerce are relatively new in systems development technologies. Customers want e-commerce, m-commerce, complex applications linked to robust databases, and Web front ends linked to legacy systems. The IT project manager must use architects and developers with backgrounds in the object-oriented world to build wireline and wireless applications. Many different resource talents are required to analyze and build a wireline e-commerce application. The resource pool consists of information and technical architects, Web masters, system quality control engineers, testers, and analysts. However, because wireless devices such as mobile phones, personal digital assistants (PDAs), and two-way pagers currently have limited display space and only allow for a simple graphic image, the team dynamics is focused on content and limited user interaction.

Potentials exist for wireless business opportunities given the growth rate of wireless technology:

- ***Complementing web-based applications by allowing users to communicate and conduct m-commerce on a potentially global level.*** In the next few years, it is projected that almost one half of business-to-consumer e-commerce will be initiated from smart phones.
- ***Offering high value within low bandwidth by focusing on time-sensitive (i.e., real-time) dynamic content such as stock prices.*** Given the relative high cost of using a mobile device as opposed to a personal computer (PC), static content may not offer the best value to customers.
- ***Developing applications that leverage tweezers versus browsers.*** Browsers on a PC can surf the Web and retrieve a large volume of information from potentially many sites at a time. Tweezers from a mobile device pluck small amounts of content (nuggets) from a specific site, which is a relatively quick and inexpensive task. Wireless devices offer either limited graphic capability or none; thus the

Wireless Communication

users experience is largely based on content. This is a potentially difficult paradigm shift from a Web page that allows a company to use logos and photos to market its brand.

Wireless Communication

Wireless communication is the marriage of voice, video, and data traffic on a single network with high speed. A project manager may exceed the estimated budget if a customer who wants to leverage wireless capabilities into a wireline e-commerce project decides to treat them as completely separate projects. Even though the client devices and mode of communication are different, the backend technologies can overlap in the areas of Web server, content provider, mail server, etc. For example, the same news supplier can be a partner to both wireline and wireless systems. If the news supplier provides extensible markup language (XML) content feeds, then the wireless system can take clippings (a summary of the whole information) while the entire feed can go to a PC.

If a project requirement is to employ an open architecture to make the system open for either third-party servers or other internal servers, then a common language becomes useful. The language to pass objects and aggregate content information has also evolved from corporations using proprietary application program interfaces (APIs) to using standards based on purposes, such as the ability to communicate with systems outside their domain. One language that is emerging is XML. This markup language uses document type definitions (DTD) and schemas to allow the receiver of a message to understand the context of what the sender is conveying. XML is a descriptive way to share concepts in a formalized fashion. If one thinks of XML as a language, then one can also consider that XML has many dialects.

IT managers should understand and prepare for the influence of the wireless information revolution by addressing the following issues:

- Which standards in communication protocols or devices are the most cost-effective while allowing the flexibility that users demand?
- What relationships should be fostered with key players in the wireless information value chain?
- How will support and security issues be treated?

Wireless services allow customers to select any local provider and deliver always-on high-speed Internet access (12 times faster than traditional dial-up modems) and competitive long-distance rates. Wireless service is an uncomplicated all-in-one package. The customer puts an antenna on his or her home, and all data and voice requirements are carried through one wireless connection. No new line, excavation or construction, or city or country permits are required. The result is high-speed Internet access with voice service via wireless communication at an unlimited distance (*The Sunday Oklahoman*, April 23, 2000, p. 2-c).

By listening to and understanding the users needs, IT managers can take advantage of the wireless revolution by offering the services that their users want. These wireless services can be access to critical business information, Internet access, integrated organization directories, or wireless meeting scheduling. The following are benefits of wireless services:

- Convenience
- Speed of transactions
- Ease of use
- Ability of information specifications
- Fast delivery
- Paperwork reduction
- Cost savings

Extensible Markup Language

- No dealing with sales representatives or catalogs
- Seeing what is being bought
- Time savings
- Short procurement cycles

Extensible Markup Language

XML is a solution for content modeling and creation of standards for content. XML helps IT system developers design application-specific vocabularies. Its popularity is based in part on human readability and ease of understanding. XML facilitates the reuse of content by storing information at an element level rather than a document level. With XML the developer can author the information once and produce it many times without additional work.

XML is also a tool for database projects. A visual designer thinks of XML in terms of client side issues such as presentation and style sheets (North, 1999). XML allows a developer to express some concepts (inheritance) in the schema so that objects can be passed from one system to another with the use of DTDs and schemas.

In XML, a project can use a DTD and schema to attach with an XML document to allow other systems to understand the message. Various standards organizations have formed, such as the World Wide Web Consortium (W3C), to define XML standards to ensure that XML can be used to exchange information with companies that are interested in the same content. The markup adds information about the quality of the data, but the different parties must agree on how to interpret the markup. Industry standards bodies, such as the Open Travel Alliance, have also been established for defining a common standard in which travel-related industries (e.g., airlines, car rental companies, and hospitalities) can communicate with one another and their customers. For example, one airline can use the same XML schema as another airline to communicate. Eventually the same schema can be proposed throughout the industry, possibly becoming a de facto standard. Given that the entire industry in this context would share the same vocabulary, it can allow some constructlike subclassing because all of the servers in the airline industry understand it. In XML, a project can use a DTD and schema to attach with an XML document to allow other systems to understand the message (www.w3c.org; www.opentravel.org).

Application of object-oriented concepts to XML would impose a definition of rules in DTDs or schemas. All interested parties should agree to and apply these rules since there is no inherent engine for XML that imposes rules. Extensible style sheet language (XSL) allows transformation from one vocabulary to another. WML is similar to HTML and is a derivative of XML. Wireless devices usually come equipped with little memory relative to a PC. WML is optimized for small devices and for wireless efficiency. The code is compiled at the gateway into and from binary that decreases latency, decreases usage costs, and increases battery life of the device. WAP-enabled telephones use a microbrowser to make a request and receive the response in WML. However, the application server may generate an XML response that is translated in the WAP server into WML, compiled into a binary format, and sent to a mobile network carrier to transmit to a user.

To illustrate the savings achieved by using WML instead of HTML, the IT manager should consider an average HTML page that has a request of 39,896 bytes and a response of 80,451 bytes, totaling 120,347 bytes. The corresponding average WML page has a request of 383 bytes and a response of 111 bytes, totaling 494 bytes.

XSL specifications define how to translate one XML document into another. Inclusive of this translation is XML formatting objects that can describe visual display, sidestepping the problems associated with plain HTML.

BlueTooth

BlueTooth is a key enabler for the mobile handset. It enables numerous devices to communicate wirelessly and form small ad-hoc wireless networks called *Piconets* (two to eight devices [slaves] with one master). BlueTooth has a low-cost transceiver chip (1 Mb) embedded into various devices, which allows communication via a low-power radio (2.4 GHz). It is supported by more than 1500 companies, including IBM, Intel, and Motorola, which are BlueTooth Special Interest Group (SIG) members.

BlueTooth technology is built into a 9 X 9 mm microchip. It broadcasts in a 10-meter radius at 720 kbps. BlueTooth requires little power and is intended for transmitting small amounts of data over short distances. BlueTooth is designed to work in a noisy radio environment using a fast acknowledgment and frequency hopping scheme to make secure links. BlueTooth hops to a new frequency after transmitting or receiving a packet to avoid interference (www.bluetooth.com). BlueTooth requires web sites to be rewritten in WML; translation of HTML is not as effective because it uses text instead of binary data.

Handheld Device Markup Language

HDML is a proprietary language for coding web sites for wireless applications. The concept is to create a very thin client that is appropriate for wireless communication. Components that are required are a microbrowser on client devices and a server/gateway that must be licensed from Phone.com, a software development kit (SDK), and a simulator.

Information Architecture

Information architecture contains necessary data ranging from a server to a display. The user input influences this architecture for designing the system. Current Internet call centers deliver interactive customer touch points to what was originally an anonymous and passive environmentthe Internet. Solutions on the current market begin to provide the possibilitiesa holistic view of customer interactions, preferences, behavior, affordability, and ease of use. The business goals differentiate between the service and sales channels by creating a winning Internet commerce strategy with enough flexibility to evolve in this rapidly changing marketplace. Some business goals are as follows (*Internet Telephony*, 76, February 2000):

- Reduce costs by optimizing core business processes.
- Improve communications and coordinate the efforts across business functions.
- Ensure effective customer relationship management.

Chapter 12: Distributed Object Technology

Distributed object technology provides the mechanism for linking objects in a network. With distributed objects, users can concentrate on what they need instead of how to get it. This chapter covers environments, common object request broker architecture (CORBA), the common object model, and the distributed system object model.

Distributed Object Environment

The distributed object environment (DOE) is a mechanism for objects to access one another's public state and export functionality. CORBA is technically superior to the DOE, but many vendors claim that the DOE is superior when based on Java.

The DOE is useful in client/server development and Internet applications. Developers put together complex client/server systems simply by assembling and extending reusable distributed objects. Distributed objects can be altered without changing their core functionality. Distributed objects allow the developer to specify self-defined and strongly typed entities. Thus developers can mix and match reusable assets even if the assets reside across the network.

Intranets, Extranets, and the Internet have contributed to the growth of object technology by requiring the need to better manage object distribution throughout the enterprise. An Intranet is an electronic business (e-business) application within an organization. An Extranet is an e-business application deployed within the larger community of an organization and includes its suppliers, vendors, and contractors. Connecting to the Internet makes the information it contains accessible from anywhere in the world using ubiquitous browser software. Intranets, Extranets, and the Internet require the management of distributed transactions regardless of platform, language, and scalability. The World Wide Web (WWW) is a highly advanced distributed object system. Figure 12-1 shows a graphic view of the DOE.

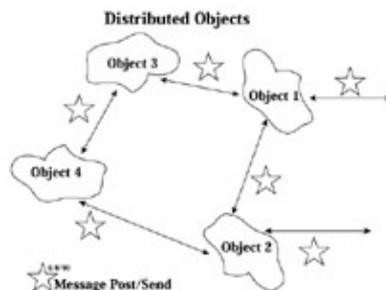


Figure 12-1: A sample of distributed object environment

The integration of the Web with CORBA or the common object model (COM) provides an excellent opportunity for distributed object technology. CORBA and COM are server-centric architectures that place the state and behavior of objects in server processes called *shared libraries* or *repositories*. Clients can send requests to objects that logically reside elsewhere. Thus the DOE is an environment that allows objects to be distributed across a heterogeneous network and allows each computing component to interoperate as a whole. When objects are distributed across a network, clients can be servers, and conversely, servers can be clients.

Distributed Computing Environment

The distributed computing environment (DCE) combines object-oriented concepts with client/server technology to produce a framework for building modular and scalable distributed applications at a relative high level of abstraction. Distributed object technology provides the means to do so by enabling access to

Common Object Request Broker Architecture

applications and data wherever servers are located. The DCE provides a standardized environment for distributed applications across platforms. The DCE is only a foundation.

In the DCE, hardware, software, and networking work together in a system. The developer prepares a graphic model of a system, reuses well-tested components, and identifies interfaces. The integration of a system saves time and money. The DCE is more cost-effective in multiple applications than in a single application.

DCE products are compatible with one another because the Open Software Foundation standardizes and licenses most of the core services. Developers can standardize DCE products on disparate platforms, and they will still work together.

Common Object Request Broker Architecture

CORBA is an object-oriented standard produced by the Object Management Group (OMG). The standard has defined many key areas of distributed computing. The CORBA standard includes mapping between an interface definition language (IDL) and C++, Java, Smalltalk, and Ada95. The OMG works with Microsoft so that CORBA and COM provide standard architecture for distributed object computing. Figure 12-2 shows a graphic view of the structure of CORBA.

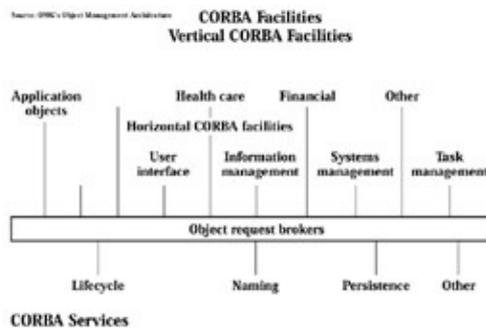


Figure 12-2: An overview of CORBA

CORBA solves interoperability problems. Each node is an object with a well-defined interface that is identified by a unique handling technique. The messages are passed between a sending object and a target object. The target object is also identified by its unique handling technique. The message format is defined in an interface known to the system. CORBA connects objects only and not applications. The OMG provides enterprise integration in object management architecture (OMA), which describes the overall architecture for distributed object computing based on an object request broker (ORB). CORBA specifies a messaging facility for DOE. CORBA consists of the following four major parts:

1. **Application objects.** The developer creates application objects specific to the applications. These exist in support of a particular application. The developer defines these objects using IDL so that the objects can communicate with other CORBA-compliant ORBs.
2. **ORB.** The ORB is the center of all activities. The ORB is middleware that establishes client/server relationships between objects. Using an ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for finding an object that can implement the request, pass in the parameters, invoke its method, and return the result. The client does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of an objects interface. In so doing, the ORB provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems.

CORBA Benefits

The ORB is a piece of infrastructure technology that helps in the development of CORBA objects and arranges for objects to access one another at run time as defined by the CORBA specification. The ORB is an engine that can communicate with other local or remote objects by using a common interface and network protocol. An ORB can make requests to other ORBs, and they can also process responses. All of these processes happen behind the scenes, hidden from the user and the client/server applications.

The ORB ensures portability and interoperability of all application objects that exist on various connected computers. In fielding typical client and server applications, developers use their own design or a recognized standard to define the protocol to be used between the devices. The protocol definition depends on the implementation language, network transport, and several other factors. The ORB simplifies this process. With an ORB, the protocol is defined through the application interfaces via a single implementation language-independent specification, the IDL.

The ORB also provides flexibility by letting programmers choose the most appropriate operating system, execution environment, and programming language for each component of a system under construction. The ORB allows the integration of existing components. In an ORB-based solution, developers model the legacy component using the same IDL that they use for creating new objects. They then write a wrapper code that translates between the standardized bus and the legacy interfaces.

3. **CORBA facilities** provide services to applications and consist of two subparts: vertical and horizontal. These are a collection of services that relate more to clients than to the server. The vertical CORBA facilities define higher-level objects and interfaces that include domain-specific vertical services. The horizontal CORBA facilities are domain-independent horizontal services.
4. **CORBA services** provide a group of services that use object interfaces to define interfaces for general purpose supporting services. They provide a base set of services encapsulated within the ORB. The object services augment the base functionality of the ORB.

CORBA Benefits

- CORBA allows applications to communicate with one another no matter where they are located or who has designed them.
- The entire system is self-describing.
- The specification of a service is always separated from the implementation.
- CORBA 2.0 provides enough detail, especially considering the object-to-object communications links, to enable two ORBs from two or more vendors to work together.
- The user discovers objects and reuses them.
- The CORBA concept relates to domain and subdomain relationships.
- The CORBA specification includes how objects are defined, created, dispatched, and invoked and how they communicate with one another.

Interface Definition Language

The IDL defines interfaces to objects and defines the interface of each object in a unified way. The ORB interfaces are defined via an IDL. The interface definition specifies the operations that the objects are prepared to perform, the input and output parameters that they require, and any exceptions that may be generated along the way. Remote objects view a CORBA object purely in terms of this interface. The IDL is accessible to objects written in a cross-platform communications architecture in any language. The CORBA architecture separates the interface, written in OMG IDL, from the implementation, which must be written in a programming language.

CORBA Architecture

To the client or user, the interface represents a promise that when the client sends a proper message to the interface, the response will come back. To the object implement, the interface represents an obligation that he or she must implement, in some programming language, all of the operations specified in the interface.

CORBA Architecture

In addition to the ORB and IDL, the major components of CORBA are as follows:

- The interface repository contains all IDL definitions that describe the attributes, operations, user-defined types, and exceptions of the server objects.
- The basic object adapter (BOA) represents the interface between the ORB and the server applications. It dispatches objects that the server applications maintain and exchanges messages with the server.
- The static invocation interface (SII) is a stub-based interface used by client programs to invoke service on application objects.
- The dynamic invocation interface (DII) is a generic interface that does not require stubs but rather supports dynamic construction of object invocations by the client program at run time.
- The object reference is the value that unambiguously identifies an object. Object references are never reused to identify another object.

Common Object Model

The COM is a part of Microsofts Windows family of operating systems. The COM defines a set of base classes and a mechanism for constructing and dynamically linking objects. The COM provides the underlying object model for components. All COM objects are required to implement behavior that allows for the software component plug and play. Microsoft engineered the COM with distribution in mind, and it initially supported communications on a single machine.

The COM provides a framework for development and deployment of software components. The developers capture abstractions as component interfaces and then provide binary classes that implement those interfaces. The COM includes an IDL derived from the open group DCE IDL.

The COM and CORBA both provide encapsulation by allowing client access to an object only through the objects interface. The COM and CORBA provide polymorphism, the ability for different kinds of objects to respond approximately to the same message. The COM and CORBA also support inheritance theoretically, but in reality the COM does not directly implement inheritance. The COM is a part of the framework of object linking and embedding (OLE) and ActiveX.

Distributed COM

The distributed common object model (DCOM) is a Microsoft object broker and is part of Microsoft Windows NT 4.0. The DCOM is an extension of the COM, extending the programming model introduced by the COM to work across the network. The DCOM provides communication across networks using remote procedures call (RPC). The future version of the DCOM is going to support other protocols. These extensions include improved location and packaging transparency, additional threading modeling, new security options, and additional administration capabilities.

Clients can access DCOM objects on remote machines without specific coding for the network. Alternatively, clients can specify the remote host that should be accessed when creating a new DCOM object.

ActiveX

The DCOM uses the same mechanism as CORBA IDL but with a different nameobject definition language (ODL). The DCOM supports distributed application development and provides directory, security, transaction, and processing support. With its mechanisms for placement of parameters, client proxies, and server stubs, the DCOM distributes COM object infrastructure to support client requests made to objects on remote computers as shown in Figure 12–3.

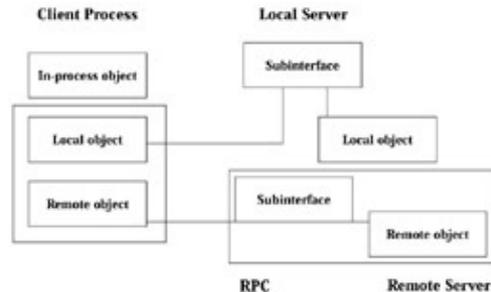


Figure 12–3: DCOM client server application overview

The DCOM is an OLE–enabled infrastructure for distributed development. The DCOM provides developers and users with access to data from other OLE–enabled applications. Linked applications can share data, whereas embedded applications actually commit their data to applications that access them. The basic OLE concept formed the foundation of the COM, which specifies how to build COM and OLE components and distributed objects, including the application program interfaces (APIs) and protocols that link COM/OLE components and distributed objects. The COM also specifies naming conventions, security, and directory services for object–to–object communications. The COM defines how components are built and interact; the DCOM defines distribution and communications with COM–enabled objects on the network.

The DCOM itself is not a development environment but rather a means of distributing application services. Developers can build servers for distribution using any COM–enabled development tool for ActiveX. Once the ActiveX exists, the DCOM takes over. A DCOM server can reside anywhere. Developers need only reference the server in the client applications, and the DCOM can find the correct resource for the client.

ActiveX

ActiveX is compatible with Microsoft Internet Explorer 3.0 or later versions that Microsoft offers free of charge. Microsoft designed ActiveX as a simplification of the COM to support linking of objects and applications across the network. ActiveX is widely discussed as a means of dynamically loading applets across the network, but it actually supports client or server side objects. ActiveX can be used interchangeably with browsers or standalone applications.

The most widely used ActiveX objects are so–called controls that implement user interface components; however, an ActiveX object need not have a user interface. An ActiveX control is a COM server in process. ActiveX components exist as precompiled binary objects suitable only for a specific platform.

CORBA 2.0 and Interoperability

All CORBA 2.0compliant products provide interoperability among one another using Internet inter–ORB protocol (IIOP). IIOP provides standardized protocol for all CORBA–compliant ORBs. This enables object development using ORBs from different vendors to interoperate transparently. The IIOP communication reduces network traffic while expanding the services that can be offered by a Web browser. The OMG has adopted a formal specification for COM/CORBA *interworking*. An OMG COM/CORBA committee created the term interworking to distinguish interoperability between technologies from inter–ORB interoperability.

Distributed System Object Model

Building distributed applications requires a proven infrastructure that scales to handle the diversity of the enterprise or the Internet. CORBA provides that infrastructure, allowing the developer to build standards-based systems from components implemented in a variety of languages and running on many different platforms. CORBA guarantees interoperability between these components through use of a standardized wire-protocol (IIOP). The IIOP defines a common data representation and a set of request and reply messages. The ORB from a single vendor has no problem communicating from clients to servers across a network. The ORB from different vendors can also interoperate with IIOP.

On the Active platform, the DCOM is the plumbing and wiring that connect COM objects into distributed applications. The DCOM acts as part of the operating systems infrastructure, allowing ActiveX and Active Server objects to find each other over a network.

Distributed System Object Model

The distributed system object model (DSOM) is an IBM product and is an extension of the system object model (SOM). The SOM is an object-oriented technology for building, packaging, and manipulating binary class libraries. The SOM is language neutral. The SOM preserves the object-oriented characteristics of encapsulation, inheritance, and polymorphism. Major benefits of SOM are that the changes to SOM class do not require source code changes in client programs and those client programs need not be recompiled. The SOM conforms the OMG CORBA standard. With the SOM, class implementers describe the interface for a class of object in CORBA IDL.

CORBA 3.0

CORBA 3.0 will include IDL mappings for developers writing in COBOL, Java, or a fourth-generation language, such as Sybase division Powersofts PowerBuilder, to access and generate CORBA objects without wrestling with the complex IDL. A CORBA component model would allow developers to create components visually on a project palette and access methods and properties more easily than the coding-intensive approach required by IDL. The component model will be based largely on JavaBeans. JavaScript will be among the scripting languages considered according to the chairman of the OMG.

The OMG plans to bolster its forthcoming CORBA 3.0 specification with a DOM for building components that is designed to head off DCOM. The CORBA 3.0 addition, tentatively called *CORBABeans*, aims to enable the creation of lightweight, reusable, graphic-based components that can be distributed across an enterprise. The CORBA 3.0 specification will provide mapping from Java to C++ objects and hide CORBA IDL to ease programming. The OMG has proposed three stipulations to the CORBABeans specification:

1. The specification should equally address the needs of the Java community and the C++ community.
2. The Java-related features should be pertinent and not brought into CORBA for no apparent reason.
3. CORBA 3.0 must be compatible with earlier versions of CORBA.

Case Study 12-1

The Advanced Trading Corporation (ATC) of New York is interested in buying a special type of sugar, Brown Sugar, from the international market. To gain the competitive edge, the company used distributed object technology as shown in Figure 12-4. The ATC has many branch offices throughout the world.

Case Study 12–2: Management of a Bank Account

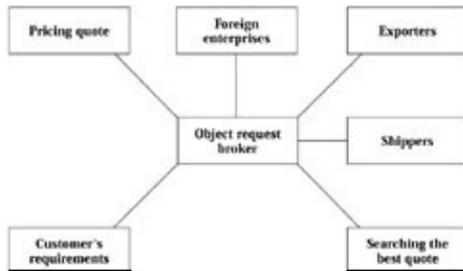


Figure 12–4: Logical flow with distributed object technology

The ATC is using C++ as the object-oriented language for their inventory system at their headquarters in New York. The system runs in a UNIX environment. The shipping system was developed by the ATC shipping agent in Singapore in the COBOL language and runs on a mainframe. The ATCs branch office in Los Angeles is monitoring current pricing on the Internet from all over the world and maintains this information in a Microsoft Excel spreadsheet. The new order processing has been developed in Smalltalk and is run on a powerful workstation by the purchasing agent at the ATC headquarters.

This application combines different hardware and software technologies. However, neither application developers nor users care about machines, locations, or programming languages. They view the world in terms of distributed objects.

Case Study 12–2: Management of a Bank Account

A user of an account object will wish to make deposits and withdrawals. An account object will also need to hold the balance of the account and perhaps the name of the accounts owner.

An example interface is as follows:

```
//IDL
```

```
interface account {
```

```
//Attributes to hold the balance and the name
```

```
//of the accounts owner.
```

```
Attribute float balance;
```

```
Readyonly attribute string owner;
```

```
//The operations defined on the interface.
```

```
Void makeDeposit (in float amount,
```

```
    • Out float newBalance);
```

```
Void makeWithdrawal (in float amount,
```

```
    ♦ Out float newBalance);
```

```
};
```

Case Study 12–3: A–Bank Information System

The account interface defines the attribute `balance` and `owner`, which are properties of an account object. The attribute `balance` may take values of type `float`, which is one of the basic types of IDLs and represents a floating point type. The attribute `owner` is of type `string` and is defined to be `readonly`.

Two operations, `makeDeposit ()` and `makeWithdrawal ()`, are provided. Each has two parameters of type `float`. Each parameter must specify the direction in which the parameters are passed. The possible parameter modes are as follows:

- ***In.*** The parameter is passed from the caller (client) to the called object.
- ***Out.*** The parameter is passed from the called object to the caller.
- ***In–out.*** The parameter is passed in both directions.

Here, `amount` is passed as an `in`-parameter to both functions, and the new balance is returned as an `out`-parameter. The parameter-passing mode must be specified for each parameter, and it is used to improve the self-documentation of an interface and help guide the code to which the IDL is subsequently translated.

Line comments are introduced with the character `//` as shown. Comments spanning more than one line are delimited by `/*` and `*/`.

Multiple IDL interfaces may be defined in a single source file, but it is common to define each interface in its own file (Orbix by IONA).

Case Study 12–3: A–Bank Information System

To remain competitive, A–Bank began offering products such as mutual funds and brokerage accounts. The customers expected to receive a single unified statement listing the balances and transactions for all of their accounts, including checking and savings accounts, mortgages, credit cards, brokerages, and retirement accounts. A–Bank realized that to meet customers demand, they needed to change from an account focus to a customer focus. This would result in customers expecting their banking activities to be structured in terms of their overall relationship with the bank rather than in terms of a single account that they may own.

However, like most banks, A–Banks computer systems were optimized for account processing rather than for providing integrated customer data. Mainframe computers managed direct deposit accounts, and a Digital

VAX/VMS system tracked mutual funds. Meanwhile, A–Bank outsourced brokerage account processing to a vendor who used a Tandem system. As a result, bank employees were required to log into several different computer systems, all with character–cell user interfaces. What should have been routine transactions for customers required bank employees to navigate three or more disparate information systems as shown in Figure 12–5. A simple question from a customer, such as *How much money do I have in all of my accounts combined?* could require an employee to go to several different systems of record, none of which offered a graphic user interface. This impeded efficient customer service.

Case Study 12-3: A-Bank Information System

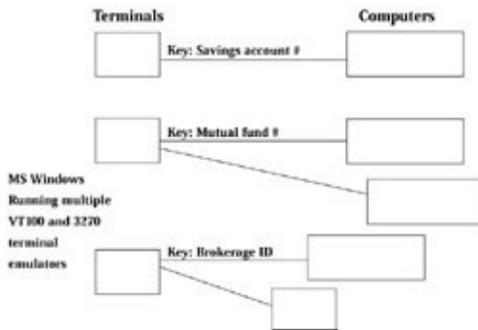


Figure 12-5. Before information access

A-Bank began to seek remedies to this problem. Reengineering of all of the banks systems of record to embrace client/server or any other technology would be neither practical nor cost-effective. Too much investment already existed in the legacy systems. A-Bank needed a way to overcome the inherent disparity between the existing systems. They needed to seamlessly integrate these systems functionality and preferably deliver the integrated result through a graphic user interface. A key objective was to deliver the integrated functionality through a native Microsoft Windows user interface.

A-Bank consulted Digital Equipment Corp. The Digital/Cushing Group Team recommended a three-tier client/server architecture to deliver a rapid solution that was production-grade quality. The Digital/Cushing Group Team also recommended the use of Digital's ORB technology to meet the goals. The ORB offers a way to construct distributed application systems using object-oriented semantics to define an application-level communication protocol between client and server programs. The selection of Digital's ORB directed the effort within A-Bank that would become the success story of the CORBA-based application development effort as shown in Figure 12-6. This technology enabled A-Bank to become the first bank to offer Internet-based access to account balances and several other innovations (Digital/Cushing Group Team; A-Bank is actually the Wells Fargo Bank).

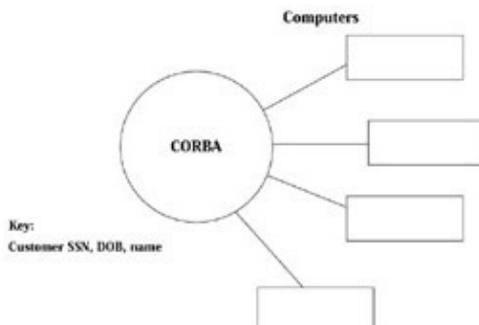


Figure 12-6: After information access

Chapter 13: Distributed Objects

This chapter presents a case study of an airline reservation system using distributed object technology in a three-tier client-server environment. The discussion surrounding the case study focuses on customer requirements and constraints, distributed object technology, three-tier client-server environments, service objects, service object replication, partitioning, load balancing, and failover. A distributed object three-tier client-server development tool called Forté is referenced throughout this chapter.

Case Study 13-1: An Airline Reservation System (OnTime Airline)

Customer Requirements and Constraints

The requirement in this case study is to create a program to handle flight reservations and accept customers complaints. This system will also provide each customer service representative with an inbox for receiving internal memos from another application. A note window (Figure 13-1) will be developed to handle special customer needs (e.g., wheel chair access, a child traveling alone). A complaint window will allow customers to register informal and formal complaints, the distinction being that a formal complaint must be in writing. The system must be able to handle up to 100 customer service representatives making airline reservations for customers who phone in requests. A legacy system is not currently in place, but a flight schedule and fare information database exists.

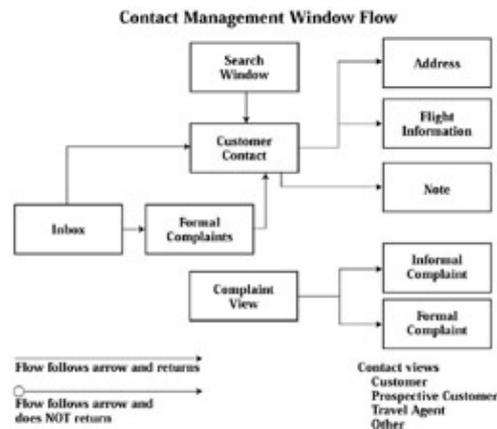


Figure 13-1: Window flow diagram

Distributed Object Technology

Distributed objects have the same properties as nondistributed objects, such as inheritance, abstraction, encapsulation, and polymorphism (overloading and overriding). However, distributed object technology allows objects to be distributed over many computers within a single environment. Objects in either a distributed or nondistributed application have a single location. A distributed application can provide access to many computers and external connections through a single integrated system. To interact with a particular object in a distributed environment, the application must interact with the computer on which the object is located; normally the object is located on the machine on which it was created. For example, if developer A builds the complaint component and developer B builds the customer contact component, then the code that developer B constructs must interact with developer A's computer to access the complaint component.

Two- and Three-Tier Client-Server Environments

An application environment is a combination of hardware and software located at development and deployment sites. A two-tier client-server environment consists of the client (tier one) and server (tier two). A simple client-server application runs on two computers: a desktop computer (client) and a database (server) as shown in Figure 13-2. For example, Forté can access multiple databases, residing on the same server and using a single database resource manager. One method to accomplish this task is by creating a file on the server, consisting of environment variables, and then accessing these environment variables as needed. Other examples consist of third-generation language applications and an application program interface (API). The phrase three-tier client-server environment refers to a client (tier one), a business layer (tier two), and a server (tier three) (Figure 13-3). Figure 13-4 illustrates the business class logic for this case study.

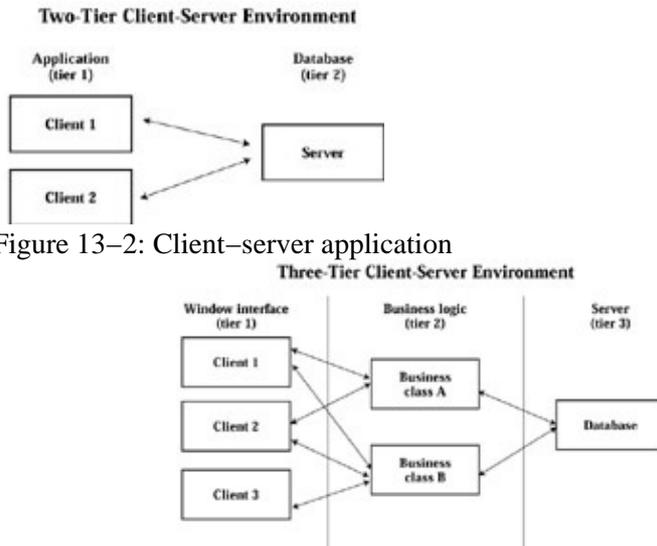


Figure 13-2: Client-server application

Figure 13-3: Various phases

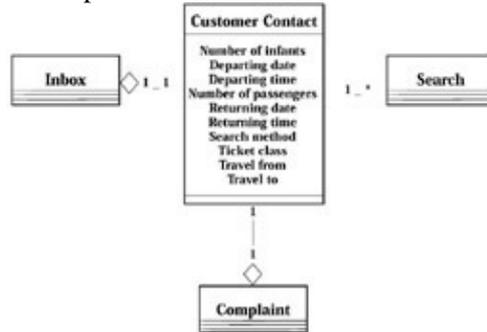


Figure 13-4: Class interaction diagram

Service Objects

A service object is a global object that can be accessed from any method within a program in the same manner as other objects. It is the means whereby an application communicates with external systems, such as databases or other applications. For example, a single database service object can be used to invoke multiple database sessions asynchronously.

Service objects can be replicated and distributed on multiple computers in the environment. For example, the customer contact window is displayed on up to 100 computers, and each computer can search the flight reservation database. Each computer must be able to register complaints on the complaint database located on the same server computer. The two objects used to interact with each database can reside on two different

Service Object Replication

computers, thus reducing network traffic.

Shared objects have an automatic locking mechanism to prevent conflicts when multiple tasks attempt to access or change an objects state. The object is locked until the transaction is complete. Thus if two customer contact representatives are viewing the availability of seats on a flight, then the one who clicks the complete button first will hold the lock until he or she completes the commit sequence.

Service Object Replication

Replication is the process of copying an object to use it for failover and load balancing asynchronously. Once the object is replicated it can be partitioned onto other computers, either client systems or servers in the environment. The objects of interest to be replicated are service objects, such as the connection to an external corporate messaging application, and shared business services, such as the flight reservation database. Server partitions can also be replicated and applied for failover and load balancing synchronously.

Partitioning

A partition consists of one or more service objects. It decouples a distributed application from the details of its deployment in a runtime environment. In the partitioning process for this case study, the airline reservation application is configured for a specific target deployment environment. The application is divided into separate logical sections called *logical partitions*. These partitions are then assigned to the computers in the specified deployment environment. Each partition is an independent process that can run on a computer. For example, an application has a client partition on the desktop that provides the graphic user interface.

Other partitions include the DBMS server, on which flight reservation and customer databases reside, that runs on a server machine. The 3GL-message system service is also partitioned onto a different computer. Forté automatically coordinates all communication between the partitions. As new hardware architectures or additional machines are added to the environment, Forté can repartition an application to take advantage of the new capabilities without requiring an application rewrite. When a Forté runtime system starts up a partition, it creates several system objects that coexist with all of the application objects within the partition. These runtime system objects support the running of each partition on its respective platform and the functioning of distributed objects as a single application. If a service object creates other objects, then they will all be located in the same partition. For example, if a service object is used to obtain flight information from a flight reservation database, then the object that was created to hold this information will be on the same partition as the service object that created it.

Load Balancing

A service object, such as the flight reservation database manager object, can be replicated any number of times for load balancing. Load balancing is the use of multiple copies of a service object, located on different computers, for simultaneous use by a multiple number of clients. A service object must be replicated to provide load balancing. That replicate can then be installed onto either the computer on which it was created or a different computer in the environment. Placement of the flight reservation database service object onto several computers reduces bottlenecks. Forté automatically provides a router partition that coordinates the requests for a service to the next available object, parallel processing, and coordinates of connections to all copies of a service object. For load balancing, parallel processing can be provided by distributing the demand for a service among several replicates of the partition that provides the service.

Failover

A service object can also be replicated any number of times for failover, which is a means to provide a backup service object to be used if the primary object fails. Failover provides built-in-fault tolerance for an application. The process for location of replicates of a service object onto different computers is the same for failover and load balancing. However, Forté will only maintain a connection to the primary service object. If that fails, then the secondary replicate will be accessed until the secondary service object will not be used. In the case of combined failover and load balancing, the router partition is replicated so that one is the primary router and the replicate is the secondary router. If the primary router partition fails, then the secondary router partition automatically becomes active.

Use Cases

Use cases are provided by the customer. The following is a sample use case for generation of a flight confirmation number.

Use Case General Information

- **Title.** Handle customer telephone call
- **Functional area.** Contact management
- **Author.** Bob Jones
- **Business contact.** Joe Smith
- **Update date.** 3/11/98
- **Business context.** A customer calls an OnTime Airline customer service representative to book a ticket for a flight
- **Actors.** Customer service representative and supervisor
- **Overview.** The actor enters all relevant information needed to book a flight

Use Case Analysis Information

- **Preconditions.** The actor wishes to reserve a seat for a customer using the customers desired flight destination, date, and time.
- **Basic course.** The actor provides the customers flight destination, date, and time to the system and then searches the system to confirm that the flight date and time exist. If the flight exists, then the system will return all available seat numbers. Based on the customers preferences, the actor reserves the desired seat number. The system issues an associated confirmation number.
- **Postconditions.** The actor has reserved a seat number for the customer.
- **Alternate course.** When the actor cross-references the flight destination, date, and time and finds no flights, then the system returns a message asking the actor to re-enter the flight information.
- **Business rules.** If the actor makes three unsuccessful attempts to reserve a seat on a flight, then he or she suggests alternative routes consisting of connecting flights.

Use Case Design Information

Use Case Design Information

Field:	Field Name:	Type:
Requirements	First name	String

Failover

Middle		String
Last name		String
Address		String
City		String
State		String
Zip		TextData
Home phone		TextData
Returning date		DateTimeData
Travel from		String
Travel to		String
Search criteria		Integer
Office phone		TextData
Total number of passengers		Integer
Number of infants		Integer
Departing date		DateTimeData
Travel class		Integer

- **Interface elements.** From the customer contact window (Figure 13–5), the actor can navigate to any of the other windows. The customer contact window offers customer information and a flight confirmation number (Figure 13–6).

The screenshot shows a window titled "Customer Contact Window" with a menu bar (File, View, Submenu, Edit) and a toolbar (New, Open, Print, Complete, Informed, Complete, Search, Customer Address List, Cancel). The main area contains several sections:

- Customer Information:** Fields for First Name, Middle Initial, Last Name, Address, City, State, Zip, Home Phone, and Work Phone.
- Number of Passenger Information:** Number of Passengers (Child) and Number of Infants (Under 2 years old).
- Date & Time Information:** Departing Date and Returning Date (both with calendar icons).
- Destination Information:** Travel From and Travel To.
- Search Criteria:** Radio buttons for "Search for best price", "Search for matching schedule", "Search only direct flights", "Travel Class" (Coach, Business, First Class), and a "Search" button.
- Confirmation #:** A field for entering a confirmation number and a "Generate" button.

Figure 13–5: Customer contact window

Discussion

Figure 13–6: Flight information

- **Preconditions.** The actor intends to search the flight reservation systems database.
- **Basic course.** The customer contact window is displayed on the screen and has a file menu bar at the top and a toolbar at the bottom of the screen. The file menu has the following items:

Save. Saves the data on the current active screen

Print. Prints the data on the current active screen; the windows default printer is used to print a bitmap image of the window

Close. Closes the current active screen

The toolbar provides short cuts to other windows or menu items. When the customer contact window is being displayed, there are validations for the following fields:

Departing date and time. This date should be the same as today or later than today.

Arriving date and time. This date should be later than or same as departing date.

Destination information. Both cities and states must exist in the database.

- **Postconditions.** The actor returns a confirmation number to the customer.
- **Alternate course.** The actor searches for flight information with the given information. The database returns more than one possible match. The actor redefines the search criteria and then selects a flight that is suitable to the customer.

Discussion

The flight reservation system application is platform independent; the same program can run on any supported client system. Thus this distributed application can be deployed in either a single environment or multiple environments; however, the effort in deploying in multiple environments consists of system administrative overhead (Forté Software, Inc., 1800 Harrison Street, Oakland, CA, 94612).

Discussion

For more information concerning Forté and their associated product line, please refer to the Forté Internet site at www.Forte.com.

Born Information Services Group, 445 East Lake Street, Suite 120, Wayzata, MN, 55391.

Chapter 14: Wireless Practical Case Study

This chapter includes a continuation of Case Study 13–1 and discusses the addition of wireless components to the airline reservation system.

Continuation of Case Study 13–1

Planning for IT project management covers coordination of management activities, phases, schedule, timeline, and staffing; delegation of duties and responsibilities; exploration of modern technology; and coordination of system software efforts in an organization. The plan focuses on integration of system hardware and software engineering and reengineering goals and principles throughout an organization. The plan also includes adoption of a software reuse approach to system development and maintenance throughout the life cycle. Changing old habits of software development via stovepipe practices is difficult. Practice of modern system development requires time to change culture and mindset. For this case study, the IT project manager covers the following steps:

- Requirements analysis
- Design
- Development of a logical model for the application
- Development of an acceptable user response time when submitting a request to the system
- Operational requirements
- Backup plan
- Definition of practitioners roles
- Life–cycle phases

Requirements Analysis

Once users, customers, and stakeholders agree on requirements, the project manager starts planning. He or she assembles resources and defines system design; prepares a schedule; and defines a phase for implementation, testing, and product delivery to the customers satisfaction.

Design

Design of the user interface for a wireless device user requires more ingenuity than that of a wireline interface because a view screen is limited to possibly four lines. As with a wireline application, the goal is to make the interface user–friendly and build navigation that is intuitive. The user should be able to focus on his or her purpose for using the device and not on the tool itself. The device should be transparent to the user, and all screens should be consistent in look and feel. The fewer the operations required by the user, the more the application will be used.

By developing user profiles, the application can be personalized so that the users experience of the application is focused on content in his or her area of interest. The profile can be specific to the application, in this case an airline reservation system application that focuses on both leisure and business travelers. Since the target audience is generally over 18 years of age, the profile information for this target audience will be different than that of an application that focuses on teenagers. Characteristics that differentiate an audience can also help define the architecture of backend servers and processes. Profiling a user for a wireless device is more challenging than for a wireline device because the design may not allow for the use of cookies and most wireless devices have a limited cache size.

Development of a Logical Model for the Application

Documentation of user activities into use cases and categorization of them (e.g., critical for application success, nice to have) allows the project team to build an application that personalizes the users experience. Critical functionality consists of activities that are required and that all users must perform. Within a use case, path activities are detailed into a main path and alternate paths, including exceptions and errors. Within each categorization, activities can be sorted according to user frequency. Based on the use cases, a use case diagram can be created to reflect a hierarchic tree of activities based on frequency. This exercise allows for a design in which the most often used functionalities are either always available or can be navigated with the least keystrokes. Bad designs are discovered during user testing.

Development of a Logical Model for the Application

Wireframes help maintain a consistent user interface. Since the display size is different from that of phones, personal digital assistants (PDAs), and two-way pagers, and hard coded keys are not universal on all mobile phones, developers must create different interfaces for each device and major brands.

For example, the back, up, and down keys are currently hard coded on mobile phones, and right and left keys may not exist. Other constraints that affect design in mobile phones consists of varying fonts on view screens, lack of graphics support, and soft-keys that may limit text size to five characters. To assist software developers in designing user interfaces, many manufacturers have made emulators available on their Internet sites. A wireless user interface consists of a deck and cards. A card consists of content, and a deck consists of one or more cards.

Definition of Acceptable User Response Time When Submitting a Request to the System

For this case study, the system that is to be made web enabled is a client/server application; thus the airline does not have a baseline and there is no hard response time requirement. Neither of the two airline upgrade projects (profiling or direct connect) has an explicit, quantitative requirement for end-user response time. However, the airline would like to maintain at least the current acceptable level of web site performance. The airline has indicated that an acceptable level of variation from the current performance is approximately 10%. Higher response times than that should be cause for project level consideration of detailed analysis of the bandwidth bottlenecks and development of performance optimization strategies. To provide a basis for measurement of response, a set of web pages, whose functionality does not radically change between now and the upgraded web site, needs to be defined as the common metric. Of the three types of end-user accessible web pages that are being developed (home pages, registration pages, and reservation pages), the latter two are completely new functions and are not available for baseline data.

Unfortunately, there is currently no hard data on response time performance of the airline site. The only partial measurement is data on the global navigation bar, which show an 8-second load on a 28.8 kbps dial-up line; however, that is only one section of any page. The airline is currently launching efforts to collect such data and measure the current response times in milliseconds from the proposed Internet service provider (ISP). The point in gathering these data is to provide a consistent and measurable test entry point. This measurement should be completed before the systems design begins.

Operational Requirements

Several operational requirements must be met, including the following:

- **Availability.** The availability of the airline generally and the software applications produced depend principally on the systems availability provided by the production system housed at the ISP. That

Backup Plan

service level of availability is 99.5% as defined in the airlines contract with the ISP.

- **Redundancy.** To ensure the availability requirements of the system, the design of the applications must accommodate redundancy through multiserver operation on each operational tier (i.e., web, application, and data). This should include the ability of the system to automatically detect and redirect operations to backup servers. Currently this includes the following:

- ◆ Load-balanced web servers
- ◆ Cold backup Direct Connect XML session server (compatibly configured servers with application software)
- ◆ The Forte software run-time environment provides its own failover redundancy
- ◆ The airlines telecommunications organization planning redundant dedicated lines
- ◆ Dual T-1s initially from the production environment to the airlines data center
- ◆ Later design decisions that may introduce additional elements, such as clustered DBMS servers, further load balancing, or load-sharing application servers, these latter elements are determined during the course of detailed design and implementation

Backup Plan

The airline has developed a backup plan to address issues of catastrophic failure of the system, the airlines Y2K Contingency and Recovery Plan. This backup plan ensures that any user-related data are backed up and archived on a regular basis, the backup operation will not interfere with the normal operation of the profiling system.

Definition of Practitioners Roles

Technical architects are responsible for the following:

- Assessing the project needs and client environment
- Providing time, complexity, and effort estimates
- Recommending, designing, and developing the system architecture
- Designing and potentially creating the application framework/foundation
- Developing standards and strategies for coding and foundation testing
- Understanding and implementing the technical architecture
- Assessing team resources
- Allocating, planning, and managing technical tasks
- Making day-to-day decisions
- Guiding the team

Technical Architect Roles

- Architect owner
- Technology expert, guru, and analyst
- Technology salesperson
- Application leader
- Technical thought leader
- Team member
- Coach and guide

Information Architect Roles

The information architect is responsible for managing information, ranging from transfer of content from a server to user interface. The users experience will be heavily influenced by how this architect designs the system. The information architect must be able to do the following:

- Write test plans, conduct usability testing, and compile results.
- Conduct usability testing and compile results.
- Understand the clients business and content, then produce a site flow document as a solution and present the solution to the client.
- Produce wireframe diagrams on projects with heavy information design requirements.
- Demonstrate a high level of communication and collaboration with clients, the design team, and other creative practices.
- Conduct the information design portion of workout sessions with clients.
- Act as team leader and mentor for other information architects.
- Understand all aspects of the creative process.
- Contribute to business development.
- Assist creative producers with management of the clients expectations.

Life–Cycle Phases

Development Environment

The purpose of the development environment is to demonstrate end–to–end functionality on the web site (Figure 14–1). The hardware and software infrastructure is sufficient to fully exercise all aspects of the target technology (Figure 14–2). However, it is not sufficient to demonstrate full–load stress testing or failover testing. A full–regression testing configuration is implemented and replicated in each subsequent environment. Software engineers writing the flight reservation system code will do all of their development in this environment and test their own modules. Thus the environment does not have to match a real–life scenario in which objects would be distributed for load balance and failover. Since the focus is to offer software developers an efficient development environment, a project manager will be responsible for obtaining large development computers and empowering as much system administration as the IT department will render and as the project development team will accept.

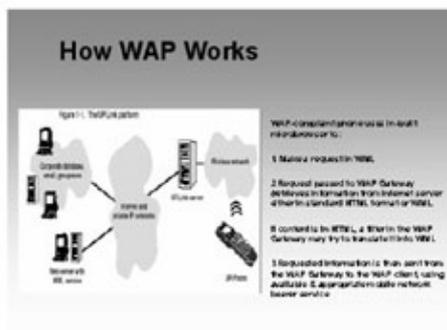


Figure 14–1: How WAP works

Testing Environment

ID	TASK NAME	DURATION	STA
1	Global Internet Presence-Infrastructure Track	1 day	Mon 6
2	Planning	1 day	Mon 6
3	Existing Inventory of WEB Sites	1 day	Mon 6
4	Identify existing WEB sites and published URL's	1 day	Mon
5	Identify key contacts for existing sites	1 day	Mon
6	Identify locations and types content currently provided	1 day	Mon
7	Identify current WEB hosting locations	1 day	Mon
8	Inventory CWT WEB hosting capabilities	1 day	Mon
9	Aggregate usage metrics from existing sites	1 day	Mon
10	Identify platform and tools in use	1 day	Mon
11	Identify Skill Inventory	1 day	Mon
12	Global Presence WEB Site	1 day	Mon 6
13	Identify WEB hosting options	1 day	Mon
14	Identify sources of static and dynamic content including 3rd party providers	1 day	Mon
15	Complete review of tools and infrastructure options	1 day	Mon
16	Forecast usage volumes based on Assumptions	1 day	Mon
17	Analysis/Design	1 day	Mon 6
18	Complete performance requirements	1 day	Mon
19	Complete analysis of database and application interactions and dependencies	1 day	Mon
20	Complete analysis of performance bottlenecks and constraints	1 day	Mon

Figure 14–2: Wireless tasks

Testing Environment

The purpose of the testing environment is to validate functionality during formal tests and perform stress testing on the hardware platform that is similar to the production environment. Full-load testing and failover testing occur during this phase.

Integration Environment

The integration environment is permanent and is used to merge the airline companys upgrade project code and data with those of other development and maintenance projects.

Staging Environment

The staging environment is either located at the ISP (with connections back to the production Forte and mainframe systems) or at the companys data center, depending on whether the IT department will accept responsibility by installing the proper infrastructure for meeting the availability requirements (first-line support). This is a permanent environment to ensure that all software components are properly configured and are ready to be placed into a production environment. This provides a brief, final sanity check at the ISP facility. No significant testing or corrections are expected to occur within this environment.

Production Environment

The production environment is located at either the companys ISP or data center. The application is located at the data center and is connected by dedicated lines. This is a permanent environment, and the current configuration must be upgraded to meet the new requirements.

With the requirements and architecture described in this chapter, a project plan can be assembled to reflect the development effort required.

Section IV: Building an IT System

Chapter List

Chapter 15: System Requirements

Chapter 16: Managing System Requirements

Chapter 17: System Design Process

Chapter 18: Software Requirements

Chapter 19: Software Design Process

Chapter 20: Software Implementation Process

Requirements are the foundation of an IT project. Failure to understand the requirements in the beginning results in an incorrect system and probably delays in delivery.

Chapter 15: System Requirements

Requirements are the foundation of an IT project. Failure to understand the requirements in the beginning results in an incorrect system and probably delays in delivery.

Requirements play a vital role in the systems development and maintenance processes. System requirements consist of hardware requirements, software requirements, and operational requirements. The IT project manager establishes a process to identify, define, elicit, and understand system requirements. The purpose is to establish a common understanding between the customers, users, stakeholders, and project manager of the requirements that will be completely addressed in the systems development. This chapter presents the importance of understanding system requirements.

System Requirement Identification

System requirement identification is important to help the customer and developers define and understand what will be involved in the system. The customer creates requirements for a specific purpose. These requirements are capabilities or conditions as stated by the customers, users, and stakeholders. Requirements can be functions, constraints, or other properties that must be provided, met, or satisfied so that the needs are filled for the systems intended users.

Requirements are the conditions that must be met for a system product to be acceptable to its customers, users, and stakeholders. Requirements can be totally new for an IT systems development project, or requirements can be for improving an existing IT system. This improvement can be possible by changing requirements in the existing system, enhancing the existing requirements, or correcting requirements to solve a problem in the existing system.

The customer and developers must understand the requirements before making a costly decision of what to build. This process involves determining, defining, and specifying requirements before analyzing them.

Requirement Determination

Requirement determination is a process that determines what is desired. Determining what is desired involves subprocesses, such as the customer defining the requirements and the system developer learning those requirements. The customer must state requirements clearly, rigorously, and precisely before proceeding to other system development phases.

The following questions are important in requirement determination:

- Who determines exactly what the requirements are?
- Does the customer know exactly what the requirements are?
- Does the IT project manager know exactly what the requirements are?
- Do the system developers know exactly what the requirements are?
- Do the system testers know exactly what the requirements are?

A combination of all of these people can determine the requirements. They form a team and work together to define, understand, and interpret requirements for a system as shown in Figure 15–1. Requirements are always unclear at the beginning. As time passes, the requirements mature because all of these members provide the necessary input. Each member of the team provides his or her input and makes the requirements clear and understandable.

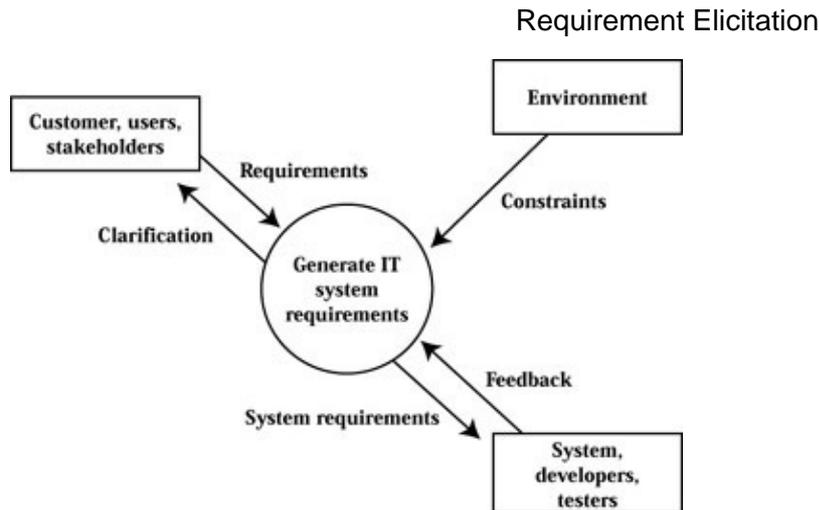


Figure 15–1: IT system requirements determination process

Requirement Elicitation

Requirement elicitation is a process performed by analysts who gather and understand information. This process involves the following factors:

- **Fact–finding.** Fact–finding uses mechanisms such as interviews, questionnaires, and observation of the operational environment of which the system will become a part.
- **Validating the customers understanding of the gathered information.** Validation involves creating a representation of elicitation results in a form that will focus attention on open issues that can be reviewed by those who provided the information. Possible representations include summary documents, usage scenarios, prototypes, and graphic models. A requirement proposal, requirement communication, and requirement definition achieve requirement elicitation. A requirement proposal outlines the need for customer requirements. The proposal requests an update that modifies or corrects an error in an existing system or develops a new system. A requirement proposal is sometimes referred to as a statement of work and states what is required from a system developer. It is an agreement that the customer and the system developer will abide by in the future. The requirement proposal may ask for the system developers ability to meet projected growth and operational changes. It can also ask for detailed costs, estimates of installation manpower, and an implementation plan. It may ask for a contractual agreement that will include responsibilities, delivery schedule, penalty clauses for missed deadlines, and explicit definitions of what constitutes the systems acceptance.
- **Communicating open issues for resolution.** Requirement communication involves the iterative elicitation and refinement of information from a variety of sources, which provides the customer with divergent perceptions of what is needed. The system analyst learns the applications and attributes of the product for delivery to the customer. The form of communication is meetings and phone conversations between the customer and the system analyst.

Frequent review of the evolving requirements by analysts who have domain expertise is essential to the convergence of this iterative process. The goal is to reach an agreement on the requirement definition. The result of the requirement definition is presented to a diverse audience, including system testers, for review and approval. Even though the customer and system analysts are frequently experts in their own areas, they are often less experienced in each others domain, which makes effective communication particularly difficult.

Requirement Definition

Requirement definition is a process. It is difficult to define requirements if they are not mature enough. The requirement may only be an idea in the customer's mind. A customer must write explicitly his or her requirements. The primary function of defining requirements is to draw blueprints and document them to eliminate potential confusion and misinterpretation. Thus the requirement definition document that the customer produces will ensure that the system developers understand the customer's requirements, needs, and objectives. This understanding and agreement will eliminate potential confusion and misinterpretation in the system's development.

Requirement definition usually includes an understanding of the environment in which the system can operate and how the system will interact with that environment. Explicit approval to proceed with requirement definition completes the elicitation process. The audience who must approve the requirements should agree that all relevant information sources have been contacted.

Importance of a Good Requirement

A good requirement is the foundation of the system. A good requirement is an agreement among the customer, users, stakeholders, and system developers. A study by the Standish Group in 1997 showed that American companies spent \$100 billion for canceled software projects. Another \$45 billion was spent on software projects that significantly exceeded their time and budget estimates. The Standish Group and other studies indicate the following top three reasons why software projects fail:

1. Requirements and specifications are incomplete.
2. Requirements and specifications are changing too often.
3. The project has a lack of user input.

A good requirement should use imperative phrases in the requirement specification. Imperative phrases command that something must be provided.

- Shall means prescribes and is used to dictate the provision of a functional capability.
- Will means describes and is used to cite things that the operational or developmental environments are to provide to the capability being specified.
- Must and must not indicate constraints. Must is often used to establish performance requirements or constraints.
- Should means suggest and is not used as an imperative in requirement specification statements.

Samples of Good, Bad, and Ugly Requirements

Good Requirements

- The organization's web site shall provide the customer and the public with accurate, timely, and relevant information on the missions and functions of the organization.
- The web site shall contain a clearly defined purpose that supports the mission of the organization and achievements.
- The web site shall be developed in accordance with the organization's management policy.
- The web site shall be maintained continuously with current data, updated at least biweekly.
- The point of contact for this requirement shall be a member of the organization's web site working

Bad Requirements

group and performs tasks as directed by the working group chairperson.

- The policy, procedure, and standard are established and delivered to the web site working group chairperson not later than 30 days before the activation of the organizations web site.
- Acceptance criteria: The web sites and pages shall be user friendly, clear, and concise. The organizations graphic representations shall be color matched and easy to understand, and the response time shall be less than 5 seconds per page. The project will be completed in 6 months. The cost allocated for this project is \$100,000, and allocated manpower is three professionals.

Bad Requirements

- The organizations web site shall provide the customers and the public with accurate, timely, and relevant information on the missions and functions of the organization.
- The web site shall contain a clearly defined purpose that supports the mission of the organization and achievements.
- The web site shall be developed in accordance with the organizations management policy.
- The web site shall be maintained continuously with current data.

Ugly Requirements

The organizations web site shall provide the customers and the public with accurate, timely, and relevant information on the missions and functions of the organization.

Requirement Types

The requirement types are functional and nonfunctional. Functional requirements are the capabilities that the system will perform. An example of a functional requirement is formatting of text. Nonfunctional requirements are the constraint on the system. Nonfunctional requirements are the performance, maintainability, safety, and reliability requirements. In addition, requirement types are categorized as follows:

- **Derived.** Derived requirements are deduced or inferred from the collection and organization of requirements into a particular systems configuration and solution.

Example: A user needs a communication system with certain operational requirements. The acquirer adds requirements concerning interoperability with other systems.

- **Explicit.** Explicit requirements are written clearly.

Example: An armored personnel carrier must be capable of sustaining a velocity of 40 kph over level ground for not less than 4 hours while carrying a full complement of troops and gear.

- **Decomposed.** Decomposed requirements clearly explain the task, action, or activity that must be accomplished to achieve a desired outcome.

Example: A fighter aircraft must be able to fly a certain distance under normal conditions before refueling. This yields requirements about fuel tank capacity, airframe configuration, expected cruising speed and altitude (which will yield further requirements for the human–system interface), aircraft weight, and so on.

- **Implied.** Implied requirements explain that the requirements are not stated but include hidden meaning.

Example: There is no benefit to be gained from buying a highly superior computer if it cannot exchange information with the computers already incorporated within the infrastructure.

Characteristics of a Good Requirement Specification

The characteristics of a good requirement are as follows:

- The customer requirements have been specified uniquely
- No duplication or overlapping of the requirements occurs.
- The requirements are needed for further analysis.
- The requirements are feasible and implementable and are not outside the capability of current technology.
- The requirements have been stated.

The characteristics of a good requirement specification are as follows:

- Unambiguous
- Complete
- Verifiable
- Consistent
- Correct
- Modifiable
- Traceable
- Testable
- Usable after development

These characteristics are interrelated. For example, requirement completeness and consistency are factors that contribute to correctness. Modifiability and testability are factors that contribute to maintainability. A requirement specification is not correct if it is incomplete or inconsistent. It is difficult to validate a requirement specification that is not understood.

Unambiguous

Only one interpretation of every stated requirement should exist. The many ambiguities in natural language must be guarded against, and ambiguity can be avoided by using a formal requirement specification language. In an object-oriented approach, a requirement is defined as follows:

- **Object.** Object-based requirements are organized in terms of real-world objects, their attributes, and the services performed by those objects.
- **Process.** Process-based approaches organize the requirements into hierarchies of functions that communicate via data flows.
- **Behavior.** Behavioral approaches describe the external behavior of the system in terms of abstract notion, mathematic functions, and state machines.

Complete

Every significant requirement that is concerned with system function, performance, attributes, interfaces, and design constraints should be included. Completeness also requires conformity with specified standards. The terms and units used should be defined, and all sections of the specification should be fully labeled. Phrases like to be determined (TBD), infinite quantity, maximum, and minimum should be avoided.

Verifiable

Requirements can be verified if they are stated in concrete or absolute terms and involve measurable quantities. All requirements except the nonfunctional requirements shall be verifiable.

Consistent

Stated requirements should not use conflicting terminology when referring to the same system element or call for different forms of logical or data structure. For example, the format of an output report may be described in one requirement as tabular but in another as textual. One requirement may state that all lights shall be red while another states that all lights shall be yellow. One requirement may specify that the program will add two inputs and another may specify that the program will multiply them. One requirement may state that B must always follow C, whereas another requires that B and C occur simultaneously.

Correct

Requirements should be defined unambiguously, clear, understandable, and in a correct logical natural language. Each requirement should be unique and consist of one meaning only. The customer has given consideration to each requirement and clarifies for hidden assumptions.

Modifiable

The specification should be structured and annotated so that changes to requirements can be made easily, completely, and consistently. The system requirement specification (SysRS) should have a coherent and easy-to-use organization with a table of contents, an index, and explicit cross-references. The same requirement should not appear in more than one place in the SysRS. Each requirement is expressed separately rather than intermixed with other requirements.

Traceable

The origin of each requirement should be clear, and the format of the specification should facilitate referencing forward and backward. Forward traceability depends on each requirement in the SysRS having a unique name and reference number. Backward traceability depends on each requirement explicitly referencing its source in earlier documents. For example, hierarchic numbers (1.2.3. 1.2.3.4, 1.2.3.4.5.6.7.8) are useful in the requirement traceability. Each requirement can be traced from its inception to its deployment in the delivered system forward and backward. The customer's requirement can be traced to its related system requirement, test requirement, designs, and program or component module. This allows impact analysis to be conducted easily any time a change to a requirement is identified or anticipated. Once the impact of the change has been identified, the customer immediately knows what components related to the changed requirement need to be changed or replaced.

Testable

All requirements in the specification should be well defined so that they can be easily tested and validated. Each requirement can have its own verification and validation criteria. This provides a high level of quality for each requirement that is propagated across the entire systems development process. It also allows the testing process to be initiated earlier in the systems development life cycle.

Usable after Development

After the system goes into operation, it should be possible to use the specification for identification of maintenance requirements and constraints.

System Requirement Specification

SysRS contains operational, technical, and logistic capabilities that are required in the IT system. The SysRS document includes the identification, organization, presentation, and modification of the requirements. The SysRS document incorporates the systems operational concepts, design constraints, and design configuration requirements into the specification. The SysRS document includes the following for the IT system:

- Desired operational capabilities
- Physical characteristics
- Performance parameters and expected values
- Interfaces with its environment
- Functional requirements
- Interactions with its environment
- Documentation requirements
- Reliability requirements
- Logistic requirements
- Constraint requirements
- Personnel requirements
- Quality requirements
- Security requirements
- Safety requirements
- Training requirements

In addition, the SysRS contains necessary characteristics and qualities of individual requirements and the set of all requirements for the system as follows:

- **Unique set.** Each requirement shall be stated only once.
- **Normalized.** Requirements shall not overlap (i.e., they shall not refer to other requirements or the capabilities of other requirements).
- **Linked set.** Explicit relationships shall be defined among individual requirements to show how the requirements are related to form a complete IT system.
- **Complete.** The SysRS shall include all of the requirements identified by the customer and those needed for the definition of the IT system.
- **Consistent.** The SysRS content shall be consistent and noncontradictory in the level of detail and style of requirement statements.
- **Bounded.** The boundaries, scope, and context for the set of requirements shall be identified.
- **Modifiable.** The SysRS shall be modifiable. Clarity and nonoverlapping requirements shall be avoided.

The objective of the SysRS is to define a capability that satisfies a customer's statement of work as shown in Figure 15–2. The purpose of the SysRS is to provide a medium for capture of what is needed and communication to all interested parties. The SysRS document serves as a contract between the customer and the system developers by defining what is to be provided and, in many instances, the manner in which it is to be produced and the technology that it incorporates. In addition, the SysRS provides a basis for most of the project's management and engineering functions, such as assessing proposed engineering changes, resolving

Usable after Development

acquirer and provider disputes, developing test requirements, writing the preliminary users manual, planning maintenance support activities, and implementing operational enhancements.

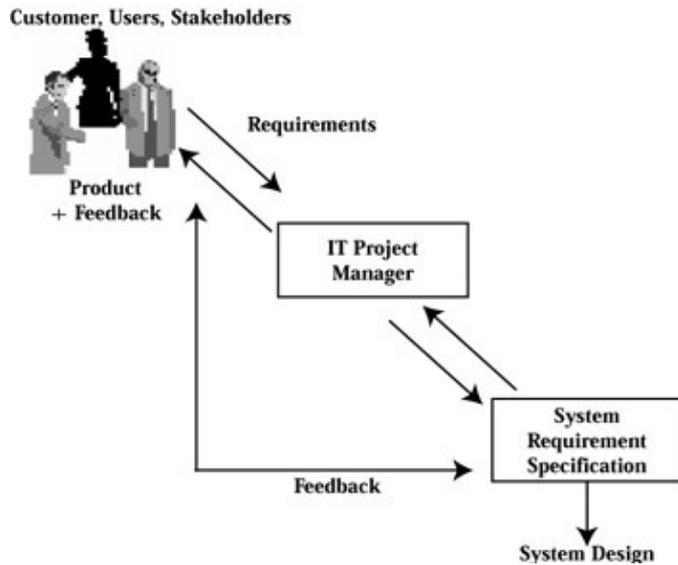


Figure 15–2: IT system requirement specification

For the SysRS to serve these purposes, it must provide a definition of the future operational and support environments in which the required capability will exist and provide a prescription for the system intended as the solution to the customers statement of work (SOW). If necessary the SysRS must also identify the methodologies and technologies that are to be used during the systems development life cycle.

The SysRS must describe the strategic and tactical environments and the systems solution within those environments. The descriptions of these environments must address elements of the operational and support situations that will impinge on the solution as they are expected to exist during the operational lifetime of the systems solution. The customer and developers must distinguish between those aspects of the environment (strategic, tactical, operational, and support) that are continuations of the existing situation and those that are expected to be delivered.

The design of the SysRS document should be based on a predetermined approach to structuring the specifications information into sections and subparagraphs (Box 15–1). Statements that are part of a single function are normally grouped together. Functions that are connected in series by output to input relationships should appear in the document in that same sequence if possible. Functions that share common inputs and outputs should be addressed within the same area of the document. If several processes must be accomplished in the same time frame, the document must tie their specifications together to make this commonality clear. Functions that are similar need to be distinguished from one another.

Box 15–1: SysRS Outline

1. Introduction
 - 1.1 Systems purpose
 - 1.2 Systems scope
2. Systems description
 - 2.1 Systems context

Usable after Development

- 2.2 Assumptions and dependencies
 - 2.3 Users characteristics
 - 2.4 Operational scenarios
 - 3. Systems interfaces
 - 4. Systems capabilities, conditions, and constraints
 - 4.1 Physical
 - 4.2 Systems performance characteristics
 - 4.3 Systems safety and security
 - 4.4 Systems operation
 - 4.4.1 Systems human factors
 - 4.4.2 Systems maintainability
 - 4.4.3 Systems reliability
 - 5. Systems external interfaces
 - 6. Systems testing scenarios
 - 7. Systems training plan
 - 8. Definitions, acronyms, and abbreviations
 - 9. References
-

Data item descriptions (DIDs) that prescribe the SysRS outline and content are attempts to solve the structural problem for a general class of documents. These DIDs only resolve issues at the highest level of organization. The DID that is to govern a particular SysRS must be tailored to fit the type of project that is to use it and the category of system that is to be developed. The parameter values shall be clearly and uniquely defined. The SysRS shall not include the values that cannot be tested, such as the following:

- TBD
- Infinity
- As a minimum, maximum
- Not limited to
- As appropriate
- Equivalent to
- If practical
- Large, small
- Many, most
- Timely
- Easy
- Normal
- Adequate

The sentences in SysRS include words and phrases that are simple and appropriate to the intent of the statement. For example, the word obscure is not used if hide is meant.

Preparation of Test Plans, Procedures, and Test Cases

Preparation of test plans, procedures, and test cases is critical in the implementation of IT systems. The requirement testing plan begins at the first phase of the systems development in which the correction of errors is least costly and the largest portion of bugs have their root cause. Identification of requirement defects at the earlier phase improves the quality of requirements. The presence of inadequate requirements is often the reason for a projects failure.

The testing starts early, when the customer initiates requirements for a system. The SysRS is the first document that introduces the systems testing. Users provide test cases to test the system; testers cannot test a system if the customer does not provide the test cases. These requirements are documented in the SysRS. The SysRS is integrated throughout the systems development life cycle.

Automated test tools are available that assess the effectiveness of testing a system. Proper implementation of a database tracks requirements at each level of decomposition and tests associated with the verification of these requirements. From this database the project can gain insight into the relationship between the test and the customers requirements.

Case Study 15–1

From humble beginnings to a multimillion dollar dietary supplement company, Mannatech lent credence to its founders belief that their network–marketing venture was something with phenomenal promise. In just 5 years the company has become a leader in what is termed the nutraceutical industry. Mannatech has a staff of roughly 300 people supporting a North American force of over 400,000 home–based associates committed to building consumer demand for Mannatechs proprietary, natural supplements. With a diverse network of nutritional experts and yearly revenues topping \$152 million, Mannatech has begun expanding into international markets.

According to Bill Councill, Mannatechs Systems and Process Engineering Manager, the companys growth can be attributed to its unique commission system. We pay eight different types of commissions, which differentiates us from other markets and certainly from other network marketing companies. Mannatechs software system manages the complex algorithms that allow for the movement of commissions throughout the associate sales structure, called the *down line*. Councill emphasizes the importance of a technologically astute software system to support such a network. It is important that our software does not have any downtime so that products are distributed promptly and commissions are paid accurately throughout the network. Our internal employees and associates will not tolerate software that does not work.

Given its fast growth and global expansion, the company has taken prudent steps toward even better management of its technological infrastructure, implementing highly dependable software applications with robust Internet capabilities. Such software capabilities enable Mannatech to support a complex commission plan for its associates and maintain a continuous flow of products and materials in a timely and accurate fashion domestically and internationally.

In many ways, development of nutritional products that would help Mannatech rise to the top of the nutraceutical market was the easy part. Developing requirements–driven applications to support the network of associates, maintaining communication between distributed groups, and meeting end–user specifications, on time and within budget, was the real challenge.

Before requirements management, Mannatech did not use a formal requirement–management process or tools for delivering software applications; rather they depended on functional specification manuals. For every

Preparation of Test Plans, Procedures, and Test Cases

month we were over time on one particular software development project, it was costing us nearly half a million dollars, explained Éoin Redmond, Vice President of Information Technology for Mannatech. We had no way of doing traceability of requirements. We had no way to easily modify or amend requirements or effectively communicate those changes. It created an atmosphere that requirements weren't important after all and that we would simply get a general idea of users requirements and figure it out as we went along.

With no formal process for defining, managing, and changing requirements, Mannatech's development teams found themselves in a code-and-go environment, bearing the burden of a project's specifications. Development time lines were simply speculative guesses, and as time went on, development costs accrued, going far beyond the projected costs. When the project was handed over to the quality assurance (QA) team, the lack of traceability between requirements and tests resulted in unfocused or incomplete testing.

Such a code-and-go approach yielded low-quality systems, which dissatisfied end-users and dampened the developers' morale. Trust in the project team had been compromised, and the end-users held the team responsible. The project team spent months attempting to resolve issues that were a direct result of unclear, undefined, and unauthorized changes to requirements.

Mannatech's project managers realized that it was time for a change. Redmond said, I am a very firm believer that if you don't control your requirements, then the rest of the project is always going to have problems. The project managers decided that a formal requirement definition and management process was the first step toward improving the quality and user acceptance of their products. They needed a way to trace requirements to specific use cases and tests to ensure a requirement-driven rather than an ad-hoc approach to development and testing.

With plans to open their first international office in Australia, requirement management became the top priority during the development of that office's software system. Mannatech required a powerful tool that would aid in communication of the software development process, providing the following:

- An intuitive, easy-to-use interface for nontechnical and technical end-users
- Communication among all departments involved in the development life cycle
- A Web-based component to form a tightly knit integration between corporate and international subsidiaries and to transmit various financials
- Mannatech's proprietary iterative and incremental development cycle, referred to as MannaFactory™, demanded tools that could support and improve their efforts without training and upkeep. From a technical standpoint, the project managers required the following:
- A robust architecture that was flexible and scalable enough to support current and future development projects
- Customizable options that were within the scope of the tool itself and through integration with custom and third-party tools for design and testing
- A framework-based tool to support the expansion of the product's functionality and scope
- A product with an established user group from which the project managers could gain insight into additional uses for the tool

The project managers needed to integrate requirements to related development and testing information. As the company moved forward with plans to achieve a successful Software Engineering Institutes (SEI) capability maturity model (CMM) review, such an integrated solution would greatly increase their ability to meet, if not exceed, SEI's high standards of quality. Mannatech sought a vendor with whom they could form a partnership and successfully achieve the goals that they had outlined.

As Mannatech evaluated various automated requirement management tools for the Australia opening, Caliber-RM™ from Technology Builders™, Inc., consistently met their standards. Caliber-RM™ offers a

System Requirement Checklist

collaborative environment for requirement management, including a group discussion feature and Web interface that is critical for involving end-users from the various departments and Mannatechs international subsidiaries in the ongoing requirements process. Through its Software Quality Management (SQM) Framework™, Caliber-RM™ provided the expansion capabilities that Mannatech needed to integrate requirement management and test management and object modeling tools.

Caliber-RM™ is integrated with Mercury Interactives TestDirector, an automated test management tool, and SELECT Software Tools (SELECT Enterprise), an object-modeling tool, both of which Mannatech already had implemented. Through these integrations, Mannatech has the ability to trace requirements to use cases and tests to ensure that the development and testing are driven by the actual requirements of the application. In addition, the SQM Framework provides the means to integrate additional tools, including problem/change request tracking, risk management, project management, and code coverage (from Technology Builders, Inc., 400 Interstate North Pkwy, Suite 1090, Atlanta, GA, 30339).

System Requirement Checklist

The IT project manager should check for the following types of requirements in the SysRS specification:

- Acceptance testing
- Data handling
- Design standards
- Maintainability
- Reliability
- Reusability
- Timing and sizing

Chapter 16: Managing System Requirements

System requirements management is an activity that spans throughout the IT systems life cycle. It relates to the changes and implementation of requirements during the systems development and maintenance. How requirements are initially gathered and stored often reveals the level of an IT organizations engineering discipline. Those that provide teams with an automated requirements environment will better support change control efforts, gain testing efficiencies and potentially reduce their future maintenance burden (Gartner Group, 1998). This chapter presents the process and tools of system requirements management.

Requirements Management Process

The requirements management process defines and maintains new and changed requirements throughout the systems life cycle. The IT project manager sets up a domain expertise analyst team to keep current, to track, and to trace the requirements throughout all phases of the IT systems development and maintenance. According to James Martin, over 50% of all project defects can be traced to the requirements management process (Figure 16–1).

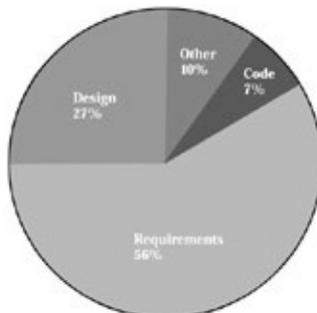


Figure 16–1: Identification of defects in system requirements

Conventional methods of requirements management capturing and communicating requirements in a document, translating requirements from one source to another, and inadequately communicating changes to requirements are not effective in the current fast-paced, distributed development environment. Often practitioners are challenged to come up with what if questions that are related to the problem that they are trying to solve. In analyzing the requirements management problem, the following what if questions often come into play.

- What if the requirement specifications are treated as a group of integrated, reusable objects instead of a static document?
- What if requirements can be captured at their source instead of being gathered and translated from one source to another?
- What if requirements and their associated attributes and representations are stored in a central repository where they can be communicated to and collaborated on by the entire organization?
- What if all users, customers, and stakeholders are electronically notified when a requirement changes?

Months or years before project completion, the ability to effectively manage requirements determines how, when, and how expensively completion will take place. Before processing a requirement, the project manager must develop a schema for the requirements management database.

Attempting to manage requirements through a document can be time consuming based on the volatility of change, and in some cases attempting to keep the document up to date can be an impossible task. Instead of collecting requirements into documents, what would happen if each requirement were treated as an object

Requirements Management Tools

with specific attributes and behaviors? Treating requirements as objects gives them the following:

- **Visibility.** Requirements can be viewed, sorted, and filtered on an individual basis as opposed to being buried in a document. This approach provides a mechanism for clearer elicitation, analysis, and communication of requirements.
- **Reusability.** Requirements can be reused from one project to the next. This approach provides for versioning of requirements within a software project that has multiple releases.
- **Testability and traceability.** These attributes are defined in Chapter 15.
- **Maintainability, safety, and security.** Each requirement can have its own change history and level of security. This provides an individual record of changes, identification of who made the change, and the reason for the change, rather than forcing users, customers, and stakeholders to review a specification document annotated with change bars.
- **Characterization.** Several pieces of data of various types can be stored for each requirement. In addition to a text-based description, each requirement can be linked to other objects, such as process models, use cases, spreadsheets, documents, and test cases.

Although organizations attempt to do a good job of requirements management, Easterbrook encountered several types of requirement errors (Figure 16–2). Each of these errors is a different issue to organizations developing a system based on the incorrect requirements. The involvement of the users, customers, and stakeholders throughout the process can help mitigate these types of errors.



Figure 16–2: Types of system requirement errors

Requirements Management Tools

Requirements management tools assist in the management of requirements and have become an important aspect of system engineering and design. Considering the size and complexity of development efforts, the use of requirements management tools has become essential. The tools that requirement managers use for automating the requirements engineering process have reduced the drudgery in maintaining a projects requirement set and added the benefit of significant error reduction. Tools also provide capabilities far beyond those obtained from text-based maintenance and processing of requirements. Requirements management tools are sophisticated and complex because the nature of the material for which they are responsible is finely detailed, time-sensitive, and highly internally dependent, and they can be continuously changing.

Many requirements management tools are available, including simple word processors, spreadsheets, relational databases, and tools designed specifically for the management of requirements, such as DOORS (Quality Systems & Software, Mt. Arlington, NJ) and RTM Requirements Traceability Management (Integrated Chipware, Inc., Reston, VA).

IT organizations that are practicing requirements management can store their requirements in Microsoft Word documents or Excel spreadsheets. Some organizations use departmental databases that may include Microsoft Access or Lotus Notes. Management of requirements in a centralized repository greatly improves the

Selection Criteria of a Good Requirements Management Tool

requirements management process. It initiates the standardization and sharing of requirements across all projects, allows requirements to be easily traced from inception through deployment and maintenance, and provides a collaboration mechanism in which threaded conversations among customers can be associated with individual requirements. Most importantly, it provides more effective and efficient impact analysis, trade-off analysis, and risk analysis of requirements.

The selection of a tool is only part of the equation. A thorough understanding of the tools capabilities and the management processes that will use the tool is necessary. The tool should not be plugged into the management processes with any thought as to the influence on the tools capabilities. Adjustments may be needed in the management processes and employment of the tool to bring about an efficient requirements management process.

The established requirements management process should have each group responsible for the data in its domain without consideration for how one data set relates to another. As a result, the requirements management tool is set up to have a class for each of the organizational elements without regard to the following:

- How the requirements are being managed in each class
- Whether the requirements are relating cleanly to requirements or tests of other classes

When requirements are stored in documents, the process of notifying the affected users, customers, and stakeholders presents two challenges:

1. The individuals making changes may not know everyone who is affected by the change.
2. Even when they do know who should be notified, they cannot always ensure that notification is accomplished in a timely manner.

Using an electronic form of notification, such as e-mail, to distribute changes ensures that notification is accomplished in a timely manner. The electronic change notification should inform the customer of who made the change, what was changed, why it was changed, and when it was changed. This approach ensures that the users, customers, and stakeholders promptly learn about all changes that affect them. When the customer combines this approach with requirement traceability, he or she can easily accomplish impact analysis of the change.

Selection Criteria of a Good Requirements Management Tool

- The technology selected must support an organizations best practices for system development. Every organization handles requirements management differently. Based on the scope and complexity of a project, the time and activities allocated to requirements management can span a wide spectrum.
- Since the project manager is ultimately responsible for all requirements, the technology must not add to an already daunting list of tasks and activities. The technology must allow the project manager to effectively control the scope of the project and easily assess the influence of changes to requirements and the project schedule.
- The technology must be easy to use. It must be easy for a variety of customers to learn and incorporate it into their routine system development activities. If the project involves business managers and end-users in the requirement management process, the tool must fit into a familiar context. If it is to be used by developers, it must be robust enough to support daily usage.
- The technology must simplify generation and distribution of specification documents. Requirements specification documents will always be needed to support the new process. Every organization has unique formats for their specification documents, and the technology should easily support generation

TBI Caliber–RM Tool

of any style of document. Distribution of documents in electronic form via the Internet is essential to the way distributed development teams work.

Based on these criteria, four proven technologies can effectively and efficiently support the reengineered requirements management process:

1. An object–oriented database that is easily customizable and provides a central and secure repository for requirements definition, analysis, and traceability
2. An Internet–based tool that provides visibility, collaboration, and immediate electronic change notification
3. An easy–to–use graphic user interface based on Microsoft Windows, Word, and Windows Explorer
4. A customizable document and report generation and electronic distribution tool

A requirements management tool that tightly integrates these four technologies provides the project team with an efficient and effective way of managing all requirements from inception through deployment.

TBI Caliber–RM Tool

TBI Caliber–RM is a collaborative, Internet–based requirements management tool that facilitates requirement definition and management throughout the IT systems development cycle. Providing a centralized requirement repository and automatic change notification, the tool enables collaboration and communication among project teams, assisting them in identifying and eliminating requirement errors earlier in the application life cycle. Caliber–RM also allows team members to compare project baselines to easily manage scope creep and identify factors that may affect schedules and budgets. The tool supports reusable requirements and ensures that project teams can build from previous experience and applications, enabling more rapid development and better use of resources. The organizations can instill discipline in their development cycle through a structured requirements management process, minimizing the cost of reworks due to requirement errors, focusing team members on the project scope, and decreasing the likelihood of project failures and overruns. The following are the main features of the TBI Caliber–RM tool:

- Requirements–driven development and testing
- Clear communication
- Shared requirements
- Internet–based nature
- Traceability throughout the life cycle
- Change management of requirements
- External references
- Automatic requirement import
- Comprehensive reporting
- Integration of the life cycle
- Flexibility to support processes

Requirements–Driven Development and Testing

Various studies have shown that roughly half of all application errors can be traced back to requirement errors and deficiencies. Thorough documentation and proper management of requirements are the keys to development of quality applications. By allowing project teams to define and document requirement data including user–defined attributes, priority, status, acceptance criteria, and traceability Caliber–RM enables earlier detection and correction of missing, contradictory, or inadequately defined requirements. The tool even recognizes terms used in multiple requirements and allows users to link those terms to a standard definition,

Clear Communication

promoting their correct usage throughout the project.

With a more complete and accurate set of requirements, project teams are able to focus on requirements within the project scope. Using Caliber–RM traceability, project managers are able to easily identify and manage scope creep, keeping the development effort on track and within budget. Through Caliber–RM test planning and management integration, quality assurance (QA) teams can quickly identify the influence that requirement changes may have on testing, enabling them to review and rerun affected tests to ensure that the application will meet end–user needs.

Clear Communication

Development of a quality application requires a considerable amount of communication among the project team, which may include business analysts, managers, development and QA teams, and customers. Providing a centralized repository for documented discussions about requirements, Caliber–RM can make communication easier and more effective for the entire project team. Team members can enter feedback on projects and requirements in Caliber–RM threaded group discussion, allowing others to review comments for better requirement revision and prioritization.

Caliber–RM also allows project managers to assign individuals to each requirement, allowing them to better manage and understand team member responsibilities. When a requirement is added, modified, and deleted, responsible individuals are automatically notified by e–mail, ensuring that everyone is aware of the current state of requirements at all times. Users also may register interest in requirements, allowing them to receive change notifications even if they are not assigned to a requirement.

Shared Requirements

Although most application requirements are unique, some requirements may be duplicated within and across projects. To minimize the effort necessary to manage such requirements in multiple locations and reduce the possibility of errors, Caliber–RM allows project teams to share requirements within and across projects. Shared requirements are listed within the requirement tree, and their descriptions are available as read–only. When shared requirements must be modified, the project team need only update them in one location, and the new data are displayed in all linked instances.

Internet–Based Nature

As project teams grow to include more than just developers and testers, they often expand geographically as well. Caliber–RM is designed to work efficiently over TCP/IP connections and modems. Transmitting only the information that each user needs for faster communication, Caliber–RM can bring distributed project teams closer together. The tool also offers a web access component that allows team members to view and comment on requirements through a Java–enabled browser such as Netscape or Microsoft Internet Explorer. This component provides remote access for individuals only needing review and minor updates, or comment capabilities.

The Caliber–RM user interface is designed for ease of use, with a requirement tree on the left similar to that of Windows Explorer and tabs containing requirement data on the right (Figure 16–3).

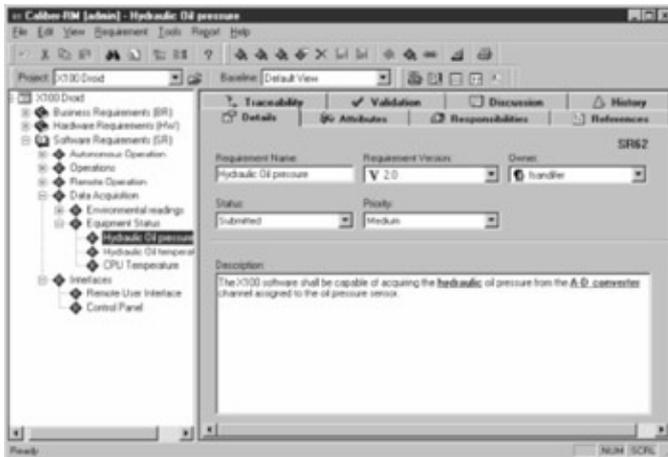


Figure 16–3: Caliber–RM user interface screen

Traceability Throughout the Life Cycle

Application requirements are rarely isolated fragments of data; rather they work together to form a cohesive unit of information. Understanding the relationships between requirements is critical to understanding the project as a whole. Caliber–RM allows traceability of requirements throughout the application life cycle, managing links between requirements to allow impact analysis and promote a better understanding of the project.

With Caliber–RM, users can see how each requirement relates to other requirements within and across projects. Through integration with third–party tools such as object modeling and test planning and management users also can follow a requirements progress through development and testing by linking requirements to use cases and tests. Using the traceability matrix, users can view all requirement relationships at once, including those between requirement and use cases or tests. When a requirement changes, Caliber–RM automatically marks all associated links as suspect to alert users of possible inconsistencies caused by the change.

Caliber–RM allows users to trace requirements to other requirements, develop entities such as use cases and business processes, and test entities such as tests and test sets (Figure 16–4). When a requirement is modified, Caliber–RM marks all associated links as suspect.

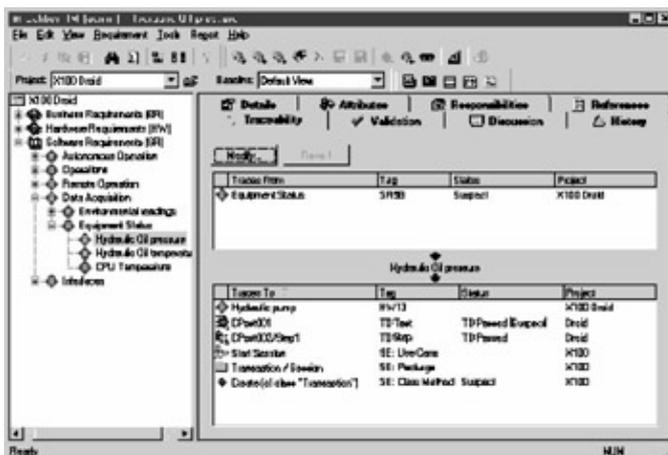


Figure 16–4: Caliber–RM requirement traceability screen

Change Management of Requirements

When developing an application, the project manager must understand how the requirements evolve throughout the development life cycle. The manager must record changes and the reasons for them to understand how and why the application has deviated from the original specifications. Caliber–RM maintains a full change history for each project and requirement, including the changes that were made, when and why they were made, and who made them. When a requirement is modified, Caliber–RM automatically increments the version number. Users can view any requirement version simply by selecting the version in a drop–down list. Users can also compare versions to better understand how the requirement has evolved.

In addition to having multiple requirement versions, project teams can use baselines to label a subset of requirements at specific versions or to capture a snapshot of the project at a point in time (Figure 16–5). Users then can view a previous baseline to see what changes have been made to the project. Users even can compare baselines to easily identify major changes and scope creep that could affect deadlines and budgets.

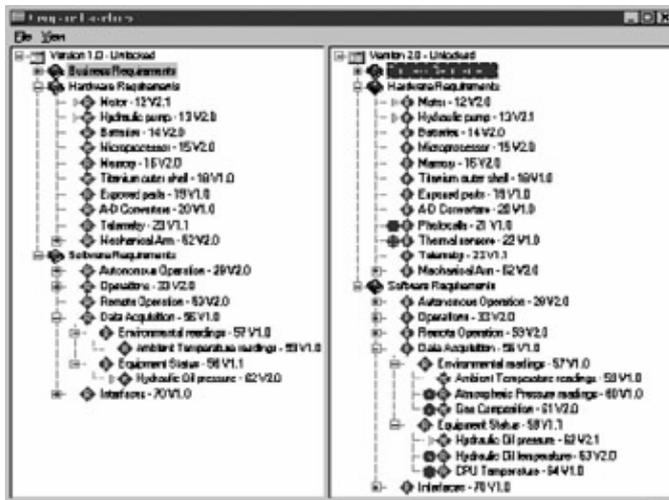


Figure 16–5: Caliber–RM project requirement baseline

Caliber–RM allows project teams to create project baselines at any point in time. Users then can compare baselines to more easily identify scope creep and understand the evolution of the project.

External References

Often, requirements need supporting information such as screen prototypes, flow diagrams, and spreadsheets to help the project team understand them completely. To make this information accessible to team members, Caliber–RM allows multiple external document references to be created for each requirement. These references can be any type of file, including graphics, spreadsheets, Word documents, and hypertext markup language (HTML). In addition, users can create uniform resource locators (URLs) and text references. Once references are created, users can double–click and open the file. Caliber–RM is closely integrated with Microsoft Word and Excel and allows users to link to text or cells within documents to specify exactly what information relates to the requirement.

Automatic Requirement Import

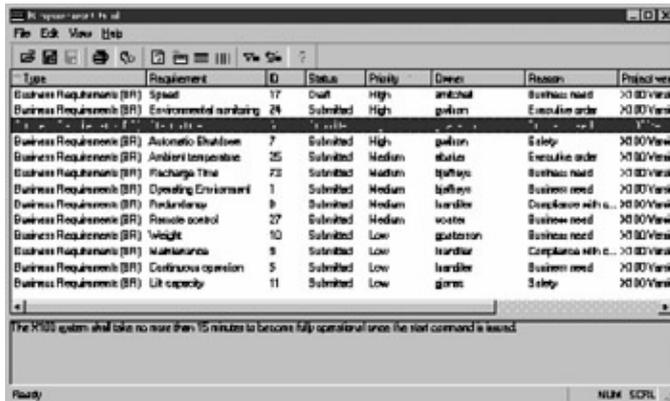
Although an automated requirements management system provides a wealth of power and flexibility for management of requirements, project teams sometimes must author requirements outside of the system, or they may receive the requirements from an outside source. For example, requirements established in a meeting can be compiled in a spreadsheet, or a customer can deliver a specifications document in Word. To

Comprehensive Reporting

enable teams to leverage existing documentation and incorporate requirements from external sources, Caliber–RM provides an automatic import capability. Using Caliber–RM Import from Word Wizard, users can specify type styles, keywords, and delimiters. The Wizard then parses the specified document and imports all requirements that match the criteria. Users can rearrange, modify, or delete imported requirements before placing them in Caliber–RM to be managed.

Comprehensive Reporting

Because each organizations development process is unique, Caliber–RM is designed for maximum flexibility. To support project teams that must produce hard copy reports, Caliber–RM provides customizable and standard reporting capabilities. The Caliber Document Factory™ allows users to create any type of customized specifications document using Word templates. These templates can use a filter and sort criteria to print a specific subset of requirements, or the templates may include all requirements. Furthermore, the Caliber–RM standard reports, requirement grid, and traceability matrix offer additional reporting options (Figure 16–6). The Caliber Document Factory allows project teams to create customized specifications documents using Word templates. Users then can generate documents as needed and use them as reports or to create project plans for development.



Type	Requirement	ID	Status	Priority	Owner	Reason	Project view
Business Requirements (BR)	Speed	17	Draft	High	ambchal	Business need	>100/View
Business Requirements (BR)	Environmental monitoring	24	Submitted	High	gusterson	Executive order	>100/View
Business Requirements (BR)	Automatic Shutdown	7	Submitted	High	gusterson	Safety	>100/View
Business Requirements (BR)	Ambient temperature	25	Submitted	Medium	ambchal	Executive order	>100/View
Business Requirements (BR)	Recharge Time	23	Submitted	Medium	gusterson	Business need	>100/View
Business Requirements (BR)	Operating Environment	1	Submitted	Medium	gusterson	Business need	>100/View
Business Requirements (BR)	Redundancy	8	Submitted	Medium	landrie	Compliance with e...	>100/View
Business Requirements (BR)	Remote control	27	Submitted	Medium	voates	Business need	>100/View
Business Requirements (BR)	Weight	10	Submitted	Low	gusterson	Business need	>100/View
Business Requirements (BR)	Maintenance	9	Submitted	Low	landrie	Compliance with e...	>100/View
Business Requirements (BR)	Continuous operation	5	Submitted	Low	landrie	Business need	>100/View
Business Requirements (BR)	Lift capacity	11	Submitted	Low	gusterson	Safety	>100/View

Figure 16–6: Caliber–RM requirement grid screen

Integration of the Life Cycle

In any application development project, the true power of requirements lies in how they are used to facilitate development and testing. Requirements that are interactively linked with development and testing objects such as business processes, use cases, and test cases are more likely to be developed and tested correctly.

Caliber–RM is supported by a framework that provides the means to link Caliber–RM with custom and third–party applications including test planning and management, object modeling, problem/change request tracking, project management, and more enabling project teams to manage applications more effectively throughout the life cycle. Caliber–RM currently supports integration with best–of–class object modeling and test planning and management tools.

Through Caliber–RM object modeling integration, users can trace requirements to business processes, use cases, classes, methods, and packages. When a requirement changes, Caliber–RM marks the links as suspect to allow users to see which development entities are affected by the change. Through the test planning and management integration, team members can trace requirements to test sets, tests, and test steps. Caliber–RM automatically displays the status of each test entity and the status of the link. Through the traceability matrix, users can easily identify requirements not linked to development or testing entities to ensure that all requirements are correctly developed and thoroughly tested. Using the Caliber–RM Test Wizard, team members can generate tests automatically, including the verification information stored with each requirement to form the basis for the test. Caliber–RM automatically generates traceability links to the created tests.

Flexibility to Support Processes

Because each application is unique, an automated requirements management system must be flexible enough to adapt to any needs. Caliber–RM support for multiple requirement types, user–defined attributes (UDAs), and easy–to–use customization features make it flexible enough to support a variety of processes and applications. Using Caliber–RM reusable requirement types, project teams can organize requirements in a manner that best supports their development process. Caliber–RM UDAs allow teams to specify what data should be captured for each of those requirement types. Caliber–RM also allows users to filter and sort requirements based on their attributes through an updatable requirement grid, enabling them to examine and easily modify subsets of requirements. Users can even create custom tabs for each requirement type or hide unused tabs. The result is an interface that supports the needs of the project team, rather than forcing the project team to be controlled by the interface.

The Caliber–RM requirement grid allows users to filter and sort requirements based on their attributes, enabling them to focus on a subset of requirements (see Figure 16–6). Users can update data for each requirement individually or for a group of requirements at once.

Applications Support

Each type of application contains its own set of rules and constraints. Caliber–RM is designed to support a variety of processes and applications, allowing project teams to customize its interface to meet the needs of their environment. In addition to supporting client/server and Internet applications, Caliber–RM also supports e–commerce, enterprise resource planning (ERP), and supply chain management applications.

E–Commerce

E–commerce applications are rapidly changing the way that companies conduct business, with the potential of a worldwide market driving tighter deadlines and a greater need for quality. E–commerce applications must be secure enough to prevent data theft, reliable enough for continuous operation, flexible enough to handle rapidly changing content, and accurate enough to ensure precise data transfer. However, too often team members see formal requirements management processes as too restrictive and time–consuming for such a rapid development environment. However, because roughly half of all application errors can be attributed to requirement errors, a requirements management process is critical to delivering the level of quality that customers demand.

Caliber–RM can help project teams develop and implement e–commerce applications by providing them with the means to detect and eliminate requirement errors and deficiencies earlier in the development life cycle and communicate rapidly changing requirements efficiently. Team members then can concentrate on the project without having to worry about requirement errors that could lead to last–minute changes and instability. The result is a higher–quality application that supports the goals of conducting business in a worldwide market.

Enterprise Resource Planning

Many organizations use ERP, or packaged applications, to support their business rather than developing their own systems. Rapid growth, mergers, and globalization make it imperative that those separate systems work together, with a minimum of effort required for integration. With its centralized requirement repository and support for distributed project teams, Caliber–RM can help organizations maintain application configuration and customization requirements in one location yet allow access by team members in many locations. Caliber–RMs user–defined attributes allow project teams to report on data specific to each module and help ensure more correct interpretation of each requirement. Caliber–RM is an efficient and effective system for

Supply Chain Management

managing the scope of ERP systems.

To facilitate development of interfaces between modules and existing customer applications, Caliber–RM provides cross–project traceability of requirements, between requirements, and between development and test entities. The Caliber–RM group discussion feature provides an electronic bulletin board for team members and end–users to make change requests for specific application requirements. Project managers then can review the requests to determine the effort, time, and cost required to implement the changes before approving them. Caliber–RM provides automated impact analysis based on traceability of requirements from inception through deployment. As future releases of ERP modules are implemented, the influence of changes to existing modules can be viewed in a traceability matrix before implementation.

Supply Chain Management

As businesses expand, product lines move into new markets and their supply chains become increasingly complex. New suppliers, additional manufacturing needs, a larger inventory to store and manage in the distribution centers, and time–to–market pressures can turn a simple supply chain into a formidable challenge practically overnight. To remain competitive, improve customer service, and enhance financial results, organizations must develop and maintain a unified supply chain, integrating all of the people, processes, and systems that are involved from beginning to end.

Caliber–RM can help organizations record and control their dynamic supply chain requirements from the overall vision to the individual applications and processes. By providing a single location for documentation and management of requirements, Caliber–RM enables project teams to collaborate on each project, minimizing tangents from the original vision and focusing team members on the established requirements. Caliber–RM's cross–project traceability enables project teams to understand how different parts of the system relate to one another, facilitating a better understanding of the system as a whole. With Caliber–RM, project teams are able to meet supply chain challenges head–on, allowing them to develop and maintain a cohesive system under dynamic conditions.

Chapter 17: System Design Process

Upon acceptance of the system requirement specification (SysRS), the customers tentatively freeze the requirements. By this time the system developers are expected to have understood these requirements, and design of the system should begin. The IT project manager establishes the system design process. The system designer develops the system architecture to understand the requirements and partitions the system requirements between software and hardware requirements.

System Design

System design begins from the baseline of system requirements. The building of a system requires identification of hardware for which the system software will be developed. The requirements for the system design will be derived from the SysRS (Figure 17–1). Identification of hardware depends on the customer, who provides the necessary tools and instruments that will be used by the system developers. This process assists the system developers so that the system is tested in accordance with the requirements. Further discussion of hardware is beyond the scope of this book.

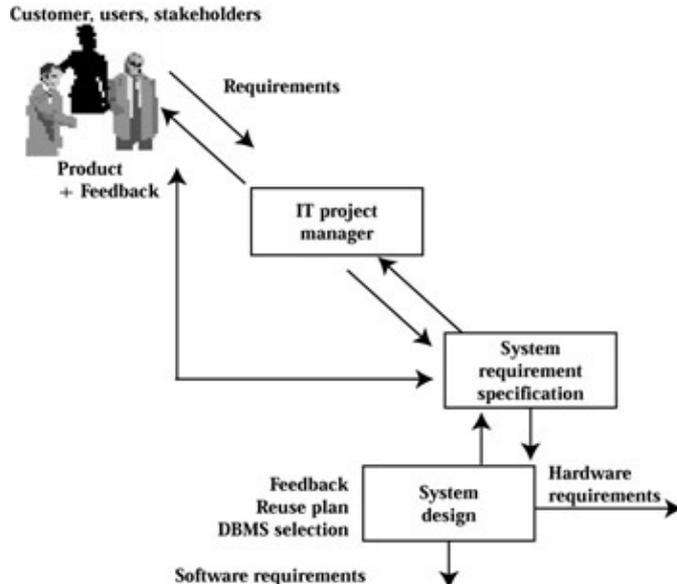


Figure 17–1: IT system design process

Identification of software and hardware depend on whether it is a new development, a reuse of the available existing system, a reengineering of the system, or a reverse engineering process. Commercial off-the-shelf (COTS) software and hardware are available that can suit the requirements. The knowledge and experience of the system or domain expertise play a major role in this selection decision. The domain expert can explore on the Internet for reuse assets. The selection must be the right assets for the right requirements.

- What the system must be capable of accomplishing
- How well the system product must perform in quantitative terms
- The environment in which system products must operate
- Human and system interface requirements
- Constraints that affect design solutions

System Design Characteristics

System design consists of requirements for hardware and software. The requirements have the following characteristics:

- Are unique and complete
- Do not overlap and are not duplicated
- Are necessary in the development of system engineering
- Are feasible and implementable
- Must not be outside the capability of current technology
- Must be consistent and stated unambiguously
- Must be evaluated
- Must have a feasible way to determine that the requirements have been satisfied and are traceable forward and backward
- Must state what will be implemented, and system design must state how to implement them

Three types of requirements are generally identified from the SysRS, which is the systems functional baseline:

- Explicit requirements are part of the functional baseline.
- Implicit requirements are implied by nature or by special knowledge of the system although not stated or expressed in the functional baseline. Implicit requirements may include requirements that are derived from a specified technical understanding of the system.
- Derived requirements are also unstated in the functional baseline but may be deduced from the structure, relationship, or nature of the system.

The system designer correctly and seamlessly maps these requirements in the system design.

Configuration Item Selection Criteria

The system designer identifies computer software configuration items (CSCIs) and hardware configuration items (HWCIs). Selection of a CSCI and HWCI is difficult. The software developer must conduct a proper analysis and select one or more CSCIs. The selection criteria depend on the following factors:

- System requirements
- Hardware
- Requirements allocation
- User acceptance
- Schedule
- Cost

The system requirements are partitioned and allocated to the lesser number of CSCIs for effective cost and time. The greater the number of CSCIs, the more work is required in the creation of the volume of documents, which further leads to more manpower needed in document checking. These schemes need more time and cost. The ideal number of selections is from one to three CSCIs.

Allocation of Requirements to CSCI

The system designer determines the best allocation of software requirements to a CSCI. The system designer

IT System Architectural Model

defines a set of software requirements for each CSCI by showing the system architecture and its interfaces. The designer explores the operational concepts of the system and discusses its functional and operational environments.

Figure 17–2 illustrates a sample system architecture, representing the allocation of system hardware and software resources. Graphically, it represents the picture of the system when built. It is the top–level model that shows all of the interfaces with external entities. The context of the system is presented graphically. If more than one CSCI has been selected, the system architecture should be shown for each CSCI for clarification. The designer should give meaningful names to the CSCI and interfaces, carry out the names throughout the system software development, and document these names in the system data dictionary. This top–level graphic becomes the foundation of the system software development.

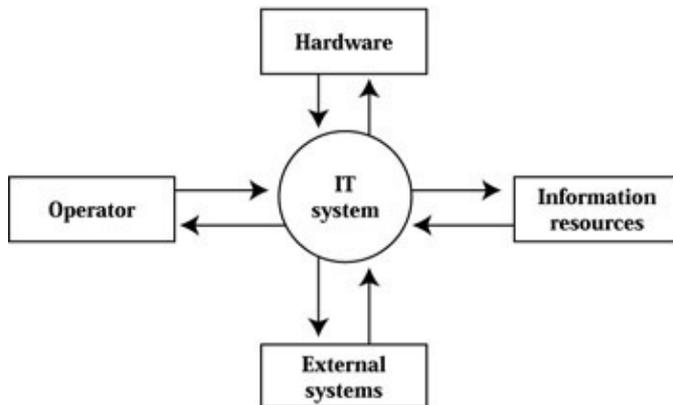
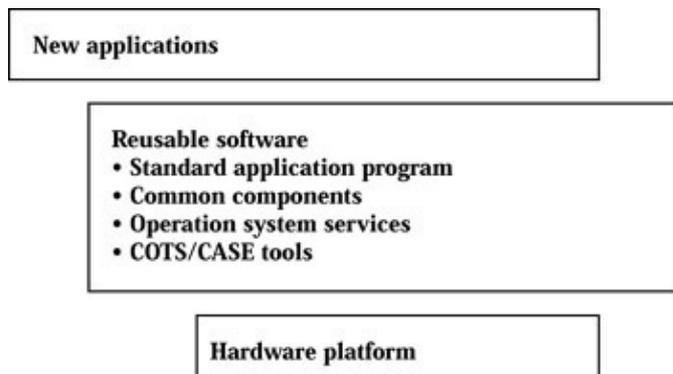


Figure 17–2: A system architecture

The designer must segregate the requirements between automated and manual. Only the automated requirements will be allocated to the CSCI for software development. The automated requirements also consist of environmental and operational requirements. All of these requirements are derived from SysRS.

IT System Architectural Model

The IT system architectural model is a generic picture of related systems in the domain. The architectural model is a plug and play, an open architecture designed around a client/server model. Figure 17–3 shows a sample of a generic domain architectural model. The model consists of a standard reusable environment, a set of standard off–the–shelf reusable components, and a set of reusable programming standards that describe how to add new functionality to the environment. The model is designed to run on hardware platforms such as personal computers (PCs) and workstations. It extends the Windows paradigm, which allows applications to coexist by providing the capability of applications to share data and services and functions at the server level.



Exploration of Reuse Assets

Figure 17–3: A sample of a generic system architecture model

Conventionally, software developers build the common function from scratch for every system. Each developer implements the functions differently, which leads to incompatibility between the related systems. Thus most of the production systems have similar functions, but they cannot easily interoperate. Surprisingly an organization is investing a great deal of time and money in rebuilding the same functions.

The concept is to construct an architecture model for building interoperable systems, an infrastructure for supporting related applications. The model facilitates an automated process for software integration and tools, provides a collection of reusable software components, and provides an approach and methodology for software reuse.

Exploration of Reuse Assets

Exploration of reuse assets for the system via the Internet is a good management technique. Reuse of well–tested components saves cost and time in the systems development. A good manager identifies the domain at early stages of the project for maximum use of reuse assets. Conducting a thorough analysis of existing systems in the organization can identify the domain or domains in the organization. Some systems may be unique, and others may be interrelated. A domain may be made up of subsets, referred to as *subdomains*. The designer should scope the domain and identify a whole domain and its subdomains.

A domain is a group or family of related systems. All systems in that domain share a set of common capabilities and/or data. The domains identification for a system is an important issue; the domain should be within the systems environment and boundary. The system developer selects a domain expert, an individual who has good knowledge of the domain and its subdomains. The domain expert explores candidates for system reuse components (i.e., subdomains) as he or she progresses in all phases of the systems development. The selection of a domain and its subdomains successfully leads to identification of system reuse components during the domain analysis.

Establishment of a System Reuse Plan

Establishment of a system reuse plan (SRP) is important to identify various phases, activities, roles, schedules, and project organizations. The purpose of an SRP is to provide guidelines and a road map to conduct day–to–day business for the system reuse center in a business organization, which consists of necessary events, actions, and processes to operate a reuse center in an organization. This includes responsibilities of various sections, practitioners, and management. The designer should only consider the plan as a guideline to follow during the development life cycle. He or she should not consider it to be untouchable. To quote a famous saying, Plans are prepared to be changed. The SRP shall contain the following:

- SRP purpose
- Various responsibilities and duties
- Operation and maintenance of the reuse center
- Customer and user participation
- Configuration management (CM)
- Quality assurance (QA) and control
- Training
- Repository system establishment and maintenance
- Reuse life–cycle phases

SRP Purpose

The purpose of the SRP is to define clear guidelines for practitioners to practice reuse in the systems development. An experienced manager establishes a reuse center and forms a team of domain experts who define the purpose, various responsibilities and duties, customer and user participation, CM of reusable assets, QA and quality control, training, repository system establishment and maintenance, and reuse life-cycle phases.

Various Responsibilities and Duties

Various responsibilities and duties relate to domain architecture, domain analysis and design, establishment and maintenance of a repository, networking, training, and management of a data review board. These include domain asset engineers, domain repository administrators, domain system integrators, domain assets quality administrators, domain assets testers, domain assets CM, and support groups.

The domain manager is an expert in the domain and management. He or she is responsible for management of the team. The domain manager ensures that the repository is functional and assets are tested in accordance with the software reuse plan. He or she is an expert in designing and maintaining the domain infrastructure architecture. He or she is responsible for reporting the progress of the reuse project to higher management. The managements commitment is important for the success of the reuse center.

The *domain engineer*, or the domain expert, conducts domain analysis and identifies and builds assets for reuse. He or she uses customer requirements to analyze and find out desirable reusable assets. The domain engineer explores other repositories on the Internet and selects from an availability of assets for the domain. His or her role is to support and satisfy the clients needs. The domain engineer also maintains domain infrastructure architecture.

The *domain analyst* performs domain analysis and identifies, collects, organizes, and represents all relevant information in a domain. This information is based on the existing systems in the domain; knowledge collected from domain experts; underlying policies, procedures, and standards; and state-of-the-art technologies.

The *domain designer* identifies a reusable asset, which is suitable to its domain. He or she has access to other repositories via the Internet and Intranet. He or she explores and researches for the right asset at the right time. The domain designer maintains a description of the available assets in the repository to be accessed by developers. Each asset has a unique identifier, and each interface is documented in the repository with proper description, protocol, constraints, and names of systems that are reusing it.

The description covers hardware, design, and programming. It also includes the processing resources and characteristics of its use. These characteristics include the following parameters:

- Memory size (amount of internal memory [absolute, spare, or both] of the computer)
- Word size (number of bits in each computer word)
- Processing speed (computer processor capacity [absolute, spare, or both])
- Character set standard
- Instruction set architecture
- Interrupt capabilities of the hardware
- Direct memory access
- Channel requirements
- Auxiliary storage

Operation and Maintenance of the Reuse Center

- Growth capabilities of any part of the processing resource
- Diagnostic capabilities
- Allocation of pertinent processing resources to each CSCI

Operation and Maintenance of the Reuse Center

This is the responsibility of the domain manager and his or her staff. The manager delegates duties and responsibilities to each member of the staff to perform certain tasks. The manager supervises the performance of his or her staff.

Customer and User Participation

This is an important factor in the success of the reuse effort since practitioners generally like to be involved in all reuse activities. Their input is important because the customer and users are the ones who use the reuse assets. They should be informed on a regular basis about the progress of the reuse center. They have proper access and services that are provided by the reuse center. There may be a fee involved for certain services.

Configuration Management

CM is the process of managing the configuration assets with a proper unique number scheme. These assets in the CM are accessible by practitioners for reuse. The CM is responsible for processing and validating submitted CM information regarding the potential, modified, or reengineered assets. All reusable assets are configuration managed. The CM personnel are the only ones with write privileges to the CM database.

Once a candidate asset is accepted and adapted for reuse, it is placed into CM control. The CM-controlled assets are changeable only if an anomaly has been discovered or the asset has been modified in use. Any asset anomalies discovered are identified with a fault report and submitted for proper analysis and disposition. The data review board for asset reuse consists of representatives from various diversified fields and systems. This board decides the course of action for the identified anomaly. It certifies and submits the corrected asset to the CM for update in the CM database. The process of building systems identifies reusable assets and retrieves them from the CM database. The CM tracks the assets that are modified and incorporated into systems via the software change order.

Quality Assurance and Control

This ensures that the reuse processes and the policy, procedures, and standards are implemented and followed. The quality section monitors, audits, and checks the correctness of various reuse documents, models, functions, and activities. They verify and validate the reuse assets.

Training

The reuse center may provide training as a service to customers and users. The training may cover topics such as repository usage, reuse assets management, reuse basics, software development with reuse, domain analysis, and design processes. The training also includes the operation and reuse of the repository. The reuse center develops and maintains a repository users manual.

Repository System Establishment and Maintenance

This is the responsibility of the software reuse center team. One of their major tasks is to select the best repository system suitable to the domains needs. The team maintains the repository to ensure that it is functional and operational at all times. The repository is easily accessible by the customers and users.

Reuse Life–Cycle Phases

These involve selection and certification of reusable assets. The phases populate the repository with the certified reusable asset with proper description and identification. The customers and users should know about the new asset. All reuse assets accepted follow the ritual for the adaptation and certification process before submitting to the CM. The reuse life–cycle phases are as follows:

- Delineation indicates how well the reusable asset is described in documentation, comments, and conformance to the policy, procedure, and standard.
- Relevance indicates how well the reusable asset matches the requirements of the functional and application domain.
- Coherence indicates the overall design quality.
- Connectivity indicates how well the reusable asset connects to other reusable assets comprising the functional and application domain.
- Inerrancy indicates how error–free the asset is and how well it has been tested.
- Mobility indicates how easily the reusable asset can be ported to new platforms and environments.

Case Study 17–1

This case study involves identification of the domain and subdomains associated with maintaining a personal checkbook. The domain is Financial Accounting (Figure 17–4). Existing subdomains must now be identified that are relevant to the task of maintaining a checkbook. Applicable subdomains include functions such as bookkeeping, accounts payable, accounts receivable, and taxable items (Figure 17–4 depicts this domain and subdomains relationship). Each subcomponent may be reused, possibly resulting in savings of money and time. In terms of software code, interfaces may have to be written if they are not yet available to glue those reusable assets.

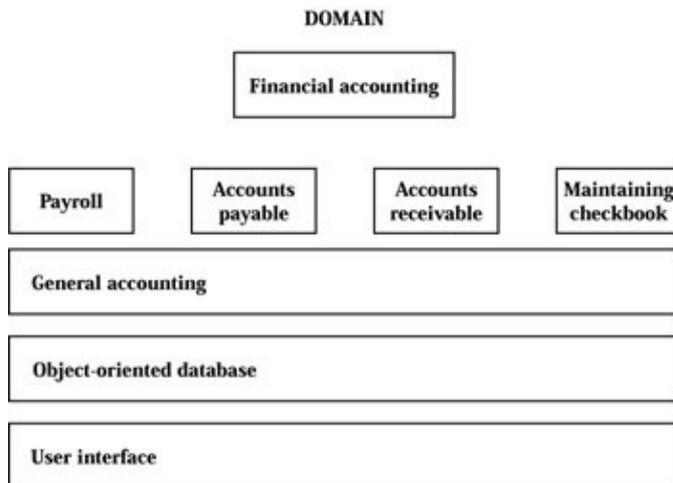


Figure 17–4: Financial accounting domain and subdomains

Selection Criteria of a Good Database Management System

Domain models represent a graphic presentation for software development. The reuse process is to understand the customers requirements and then graphically show them by logical models. Domain models determine objects and establish a relationship between other objects within the domain infrastructure.

Selection Criteria of a Good Database Management System

The selection criteria of a good database management system (DBMS) depend on application requirements, budget, and vendor support. The following factors enhance the selection criteria:

- Performance
- Ease of use
- Migration
- Data modeling capabilities

Performance

Performance of a DBMS depends on the way in which it handles the manipulation of complex data models. Manipulation of stored data should be as fast as possible and within the allocated budget and time. The DBMS operation requires a minimum overhead cost per operation. Overhead is related to pointer dereferencing (object traversal), locking, network access, and disk access.

High performance data manipulation requires that if a pointer or reference to an object is given, then the operation of obtaining data from that object must not incur undue overhead. This operation is called *dereferencing* a pointer.

Objects can be locked on a segment of the database (i.e., a page) or in a single configuration that the user defines. Only a single lock action is needed for an entire page, segment, or configuration of objects, which provides significant performance gains over other locking mechanisms.

Data caching occurs when a sequence of transactions accesses the same objects. A high probability exists that the data, which are accessed in the next transaction, will already be cached in workstation memory. Since network accesses are expensive operations, the reduction of the number of data transfers is critical to the success of a distributed DBMS. Batching objects that will be transmitted between the client and server are typically only one network request and are used when a group of referenced objects are sent into the client cache. This strategy minimizes network traffic and allows the client to proceed through a transaction without contacting the server until a transaction is committed.

Often an application uses only a portion of a database, and that portion can be stored contiguously in a small section of the database. Related objects can be clustered together in a database. This clustering increases performance because disk access time for contiguous data is faster than random access. The disks can typically read or write many sequential blocks in the same time required for disk head to move to a random location.

Ease of Use

Ease of use is in the integration between the database system and the host programming language. No inheritance exists from a special persistent object base class. Different objects of the same type may be persistent or transient within the same program. This concept dramatically improves developer productivity and code reuse. Ease of use further leads to ease of learning. In object-oriented languages, the types,

Migration

variables, and classes should be declared in the same way. Similarly, objects manipulate, access variables, set variables, and call functions, all of which should all be managed in the same manner.

No translating code is recommended in the ease of use. The programmer should not write code that translates between the disk resident representation of data and the in-memory representation that is used during execution. This is referenced to as single level storage. In contrast, developers who use the file system storage model must create a linear on-disk representation for their objects and write code that will map its in-memory representation.

Migration

Migration is a convenient way for the application and customer data to be converted into existing code, data-stored files, and libraries. Many programmers who are interested in the use of a DBMS should add persistence to existing applications that deal with transient objects and new applications that can be built from scratch. This is possible because basic data operations, such as dereferencing pointers and getting and setting data members, are semantically the same for persistent and transient objects. Variables do not need their type declarations changed when persistent objects are used.

Data Modeling Capabilities

Data modeling capabilities mean that the DBMS data model is designed so that developers can use the full power of the language for application development. With C++ and Ada, this means full support for the complete language and includes virtual functions, inheritance, polymorphism, encapsulation, and parameterized types. Additional data modeling features include the following:

1. Collections
2. Queries on collections
3. Relationships between objects
4. Versioning of objects

Collections

Collections are objects that group together other objects. Collections are abstract structures that resemble arrays in programming language or tables in a relational database. Like C++ arrays, collections store many instances of the same type or a derived type. Users may optionally describe intended usage by estimating frequencies of various operations (e.g., iteration, insertion, and removal), and the collection library will automatically select an appropriate representation. Furthermore, a policy can be associated with the collection and dictate how the representation should change in response to changes in the collections cardinality. These performance-tuning facilities simplify the developers interaction from the coding of data structures to the description of access patterns.

The collection classes for insertion, removal, and retrieval of elements of a given collection also provide methods that perform set theoretic operations, such as union and intersection, and set theoretic comparisons, such as subset. Often objects have embedded collections (e.g., a person object may contain a set of children). Collections also store all objects or a subset of all objects of the same type (e.g., all employees or managers). Such collections can be arbitrarily large. Access patterns differ greatly among applications and even over time within a single application. A single representation type will probably not provide adequate performance for all such access patterns of support for multiple representations of collections.

Queries on Collections

Queries on collections provide functionality similar to querying a relational database in that the programmer may not necessarily know how a particular object is found by following pointers. Instead, the programmer provides a query expression that selects the target object or objects that are based on the values that are contained in it or on the relationships between it and other objects. Many query languages lack the support of multiattribute, multiple condition, class extents, and distributed query functionality.

The best approach treats queries as ordinary expressions in C++ or C. A good DBMS should provide indexes that permit more efficient implementations. A query optimizer examines a variety of strategies and chooses the least expensive way for the execution of a query. The DBMS can index paths through objects and collections and not just fields directly contained in objects. In complex applications in which speed is crucial, pointers and embedded collections obviate the need for joins. Therefore a typical query will likely be over a small number of top-level collections. Selection predicates involve paths through objects and embedded collections. These paths express the same sorts of connections as those that join terms that are expressed in relational queries. Join optimization is of less concern since the path has materialized in the database with interobject references and embedded collections.

Relationships Between Objects

Relationships between objects are useful when complex objects are modeled, such as designs, parts hierarchies, documents, and multimedia information. Each relationship is composed of two or more objects that are

constrained and consistent with one another in a particular fashion. The user declares the constraints on the data members who comprise the relationship.

Versioning of Objects

Versioning of DBMS objects stores and manipulates multiple versions of an object simultaneously. This is useful for modeling many activities, such as writing a document or developing a design, in which iterative processes require experimentation with various modifications. A user can check the version of an object or group of objects, make changes, and then check the changes back into the main branch of project development. In the interim, other users continually use previous versions and therefore are not impeded by concurrency conflicts on their shared data regardless of the duration of the editing sessions that are involved.

Types of Data Modeling Capabilities

- Configurations provide a means of grouping related objects that should evolve together. An object that is composed of several objects, such as a document, may constitute a unit for versioning purposes. Configuration provides the user with specified granularity of both evolution and concurrency control. Since versioning is logical multiple representations of the same thing, a mechanism must allow a process to look at a distinct version.
- Workspaces provide a private work area so that one or more processes can view a configuration. For each application area, the system designer must determine what types of workspaces must be created. Users can then employ workspaces that selectively share their work in progress. Workspaces can inherit from other workspaces. A designer can specify that a workspace should, by default, inherit whatever is in the teams shared workspace. Individual new versions can be added as overriding of this default makes changes.
- Versioning history graphs provide a model that allows the designer and developers to visualize the evolution of versions. These graphs keep track of the different versions as the entity or entities evolve

Data Dictionary

through time. The versioning and persistence of an object are independent of type. This means that instances of any type may be versioned and that versioned and nonversioned instances can be operated on by the same user code. This can easily take an existing piece of code that has no notion of versioning and use it on versioned data.

Data Dictionary

The object, attributes, and their relationships are documented in a data dictionary. This document consists of a written description of objects, formalizes the identification of objects, forms part of the formal system specification, and separates descriptions that are written for each object. This is a live document and should start at the beginning of the system engineering and enhance throughout the systems development and maintenance phases.

System Design Document

The system design document (SysDD) contains the highest level of design information for the IT system. The SysDD describes the allocation of system requirements to CSCIs. The major outlines for the SysDD are as follows:

- Operational concepts
- System architecture
- System design features
- Processing resources
- Requirements traceability

Operational Concepts

The system developer summarizes the customers needs that must be met by the IT system. The primary mission of the system will be described. The operational environment describes the environment in which the system will be employed. The support environment describes the operational system during the production and deployment phase of the life cycle. It covers the use of multipurpose or automated test equipment and maintenance criteria.

System Architecture

System architecture describes the internal structure of the IT system. The CSCIs and HWCI's will be identified and their purpose summarized. The purpose of each external interface will be explained. Graphic system architecture will be included for clarification.

System Design Features

System design consists of the identification and description of the relationships of each HWCI, CSCI, and manual operation of the IT system. Each CSCI will be named uniquely. Each requirement from SysRS to the CSCI will be identified. Each external interface to the system that is addressed by the CSCI will be recorded as follows:

- Bits per second
- Word length

Processing Resources

- Message format
- Frequency of messages
- Priority rules
- Protocol

Any design constraints on the CSCI will be described. The internal interfaces to the system will be described, and the systems state and mode will be discussed.

Processing Resources

Processing resources cover hardware, programming, design, coding, and use characteristics of the processing resource. These characteristics include the following parameters:

- Memory size (amount of internal memory [absolute, spare, or both] of the computer)
- Word size (number of bits in each computer word)
- Processing speed (computer processor capacity [absolute, spare, or both])
- Characteristic set standard
- Instruction set architecture
- Interrupt capabilities of the hardware
- Direct memory access
- Channel requirements
- Auxiliary storage
- Growth capabilities of any part of the processing resource
- Diagnostic capabilities
- Allocation of pertinent processing resources to each CSCI
- Networking requirements
- E-mail capabilities

Requirements Traceability

Requirements traceability provides traceability of the requirements that were allocated to the HWCI, CSCI, and manual operations back to the requirements of the SysRS as follows:

**SysRS < = > CSCI requirements + HWCI requirements +
Manual operation requirements**

The traceability should be tabulated in a requirements traceability matrix. This traceability will carry out forward and backward throughout the software development documents.

System Design Checklist

The IT manager should check the following parameters in the system design phase:

- System engineering
- Operations
- Maintenance
- Testing
- Training
- Reuse
- Software

Processing Resources

- Facilities
- Personnel
- Logistic support
- Security
- Safety
- Risk
- Optimization

Chapter 18: Software Requirements

Software requirements analysis is a process that technical practitioners use to understand software requirements. Many methods are currently available to analyze customers software requirements, and many more are emerging. The IT project manager may have to select a method for requirements analysis. This chapter presents the software development plan, various modeling techniques, and software requirements analysis specification.

Software Requirements Process

The software requirements process is the starting point for logically understanding and analyzing the customers requirements (Figure 18–1). The software requirements analysis results in the definition of a complete set of functional, performance, interface, and qualification requirements for the computer software configuration item (CSCI) and subcomponents. Graphic models are used by the software developers to analyze the requirements. The goal is to present the customer with the requirements analysis in such a graphic way that less ambiguity exists than there would be in an English–language narrative. The ultimate goals are to reduce cost and increase efficiency. This process provides the following benefits:

- It minimizes risk by encouraging systematic software development and progressive definition of detailed requirements.
- It allows frequent reviews or decision checkpoints before starting each phase.
- It uses resources efficiently because parallel software development tasks can be carried out concurrently.
- It makes management planning and control easier.
- Structured analysis or object–oriented analysis is practically language and machine independent. The main concern in this phase is to understand the customers requirements clearly and assure the customer that the requirements are understood mutually and accepted by the software developers.
- It avoids excessive maintenance because all of the necessary documentation is developed as the software evolves.
- It shortens decision time for management.
- It improves communication.
- Transition from software requirements analysis to software design and software implementation phases is smooth.
- Many commercial item (CI) tools are available to assist in the software requirements analysis, design, and implementation.

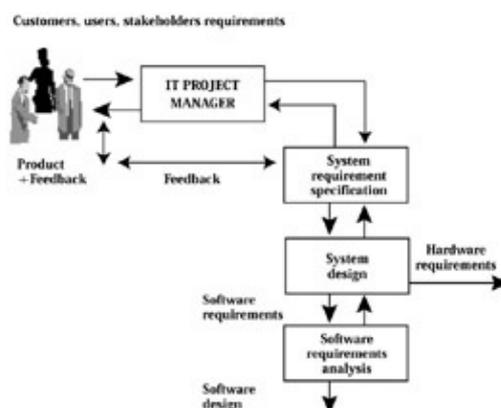


Figure 18–1: IT system design process

Software Development Plan

The software development plan describes the complete plan for development of software and how the software developers will transform the requirements for the software item into architecture. The software architecture describes its top-level structure and identifies the software components. The software developers ensure that all of the requirements for the software item are allocated to its software components and further refined to facilitate detailed design. This plan assists the customer in monitoring the procedures, management, and contract work of the organization that is performing the software development within the given time and budget.

The software developers describe their complete plan and intention for conducting software development. This includes detail of their organization, equipment, documents, facility, and training program. The plan starts with a scope, which details the systems overview and capabilities. The plan also includes the software development management plan.

The software development management plan presents the IT project organization, resources, schedule, and milestones for each activity. This includes risk management, security, safety, external interfaces, reviews (formal and informal), and all related items. The plan includes a software development relationship between the organization and resources, personnel, and software-engineering environment. Included in the plan is the use of standards and procedures while various activities are conducted. The plan explains formal and informal testing schema. It also includes software reuse, software product evaluation procedures, and tools. The plan covers configuration management (CM) and quality management factors in detail and covers the software product acceptance plan.

The plan includes the process of developing satisfactory software that may sometimes require more than one method. The IT project manager may have to select a method to be used for requirements analysis and another method for software design. The plan refers the computer language for coding logical instructions that will be used in software implementation. The plan explains the selection of a software life-cycle model.

Software Life-Cycle Model

The software life-cycle model encompasses many concepts throughout the software development. These concept phases are software requirements analysis, software design, software coding, software testing, and software maintenance. Software requirements may change with time.

Thus the life cycle provides a way of looking at various phases of software development for a system. The phases assist in going through various aspects of software development into a common language, which may be graphic, so that the effects of changes to the development process can be determined. A software life-cycle model is either a prescriptive or descriptive characterization of software evolution.

The level of detail that is supplied by the method is prescriptive in that a direction is provided on how the various activities will be accomplished. A prescriptive life-cycle model is developed more easily than a descriptive model. Many software development details can be ignored, glossed over, or generalized. This should raise concern for the relative validity and robustness of such life-cycle models when different kinds of application systems are being developed in different kinds of development settings.

A descriptive life-cycle model characterizes how software systems are actually developed. Articulation is less common and more difficult because one must observe or collect data throughout the development of a software system. This development period is an elapsed time that is usually measured in years. Also, descriptive models are specific to the systems that are observed and only generalized through systematic

analysis.

The following are types of software life-cycle models:

- Grand design
- Incremental
- Evolutionary
- Waterfall
- Prototyping and simulation
- Assembling reusable components
- Spiral
- Operational
- Transformational
- Star process
- Domain prototype
- Domain functional

Grand Design Model

The grand design model is essentially a once-through model used to determine user needs, define requirements, design the system, develop the system, test, fix, and deliver. All of the requirements are defined first.

Incremental Model

The incremental model determines the customers needs, defines the software requirements, and performs the rest of the development in a sequence of builds. The first build incorporates part of the planned capabilities, the next build adds more capabilities, and so on until the system is complete. Multiple developments are permissible in this model. All of the requirements can be defined first. Field interim products may be possible in this model.

The incremental model follows the pattern of increments of an initial subset of the system and is fully developed at that level. Then, in successive steps, more elaborate versions are built upon the previous ones. The architectural design of the total system is envisioned in the first step, but the system is implemented by this successive elaboration. The software is developed in increments (Figure 18–2), which represent degrees of functional capability. Software is built in small manageable increments. Each increment adds a new function to the system.

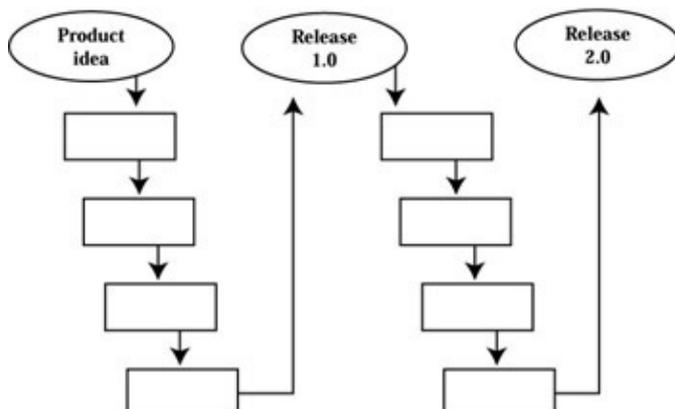


Figure 18–2: A sample of the incremental model

Evolutionary Model

The strength of incremental development is that the increments of functional capability are much more easily understood and tested at a level. Use of successive increments provides a way that incorporates the customers experience into a refined product in a much less expensive way than doing all at one time.

This model combines the classic software life cycle with iterative enhancement at the level of software development. It also provides a way to periodically distribute software maintenance updates and services to scattered users. This is a popular model of software in the computer industry.

Evolutionary Model

The evolutionary model also develops a system in builds, but it differs from the incremental model in that it acknowledges that the user need is not fully understood and all requirements cannot be defined up front. User needs and system requirements are partially defined up front, and then they are defined in each succeeding build. Multiple development cycles are permissible, and field interim products are possible.

Waterfall Model

W. Royce first introduced the classic waterfall model in 1970. The waterfall model follows the pattern of waterfall for software development. Various phases of software development, such as software requirements analysis, software design, and software implementation, precede one after another in that sequence. This software evolution proceeds through an orderly sequence of transition from one phase to the next in linear order. The model divides the software process into the following phases (Figure 18–3):

- Analysis
- Design
- Implementation
- Testing

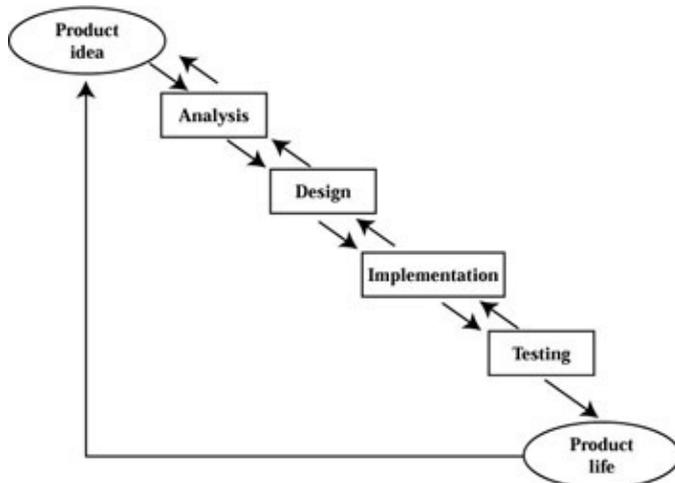


Figure 18–3: A sample of the waterfall model

Each phase is conceived in terms of input, process, and output. The software process proceeds through a sequence of steps with iterative interaction between phases, which are confined to successive steps.

Deliverable software is usually produced and then used as input into the next phase.

In reality a system or software development is never so clean. As shown in Figure 18–3, feedback loops exist between phases. Because of these feedback loops, the deliverable products at each phase need reviews and changes before their acceptance. The design phase may be subdivided into many phases with an increase in

Prototype and Simulation Model

detail.

The waterfall model derives its strength from its classic steps, which are taken successively. It provides a structured template for software development. This model is not popular among object-oriented software developers because more interactions are needed between nonsuccessive steps. Identification of all of the requirements for object-oriented software is not an easy job.

The waterfall model has been useful and has helped structure and manage software development projects that are large, medium, and small in organizational settings.

The waterfall model is the phased model that partitions the software process into a sequence of distinct phases that follow each other. Each of these phases is well defined for a specific project in terms of its inputs, outputs, activities, resources, and criteria before transit to the next phase. The waterfall model expects that each phase be completed before the next one is started, as in the top-down concept (Figure 18-3). This model is well suited for well-defined and understood requirements. In practice, the model is not so clean and has many iterations of feedback in each phase from other phases. The drawback is that the distinct phases are unrealistic in practice.

Prototype and Simulation Model

The prototype and simulation model advocates the early development of components that represent the eventual system. Often these components represent the users interface to the system. A skeletal implementation of this interface is developed with the intent that an opportunity would provide feedback from the software user before the final system is specified and designed. Clarification of the user interface is one goal, but prototyping may also be employed as a concept within the context of another model. In this case, the second model of the software process may regard prototyping as but one component of the process that will be used for clarification of the behavior of the system at an early point in the development.

Prototyping technologies usually accept some form of software functional specifications as input, which in turn are simulated, analyzed, or directly executed. As such, these technologies let the designer initially skip or gloss over software design activities. In turn, these technologies can allow rapid construction of primitive versions of software systems that users can evaluate. These user evaluations can then be incorporated as feedback, which refines the emerging system specifications and designs. Depending on the prototyping technology, the complete working system can be developed through a continual process of revising and refining of the input specifications.

This model has an advantage in that it always provides a working version of the developing system while software design and testing activities are redefined for input of specification refinement and execution.

Prototyping is one of the variations on the waterfall model (Figure 18-4). This model is useful for quick results and is without proper documents to show the concept of proof for the system requirements.

Assembling Reusable Components Model

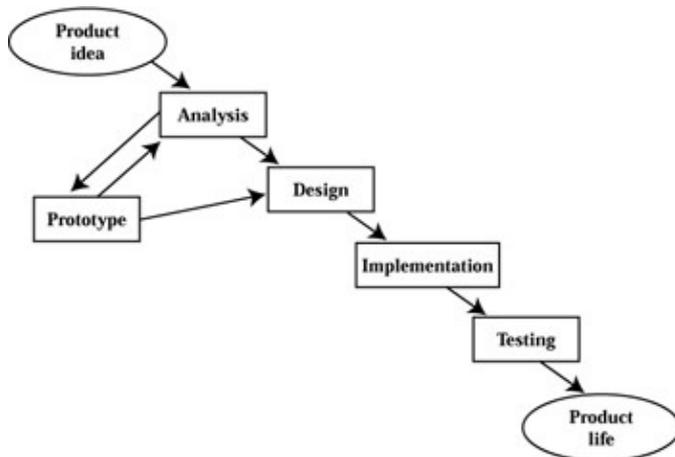


Figure 18–4: A sample of the rapid prototyping model

Assembling Reusable Components Model

The basic approach of reusability configures preexisting software components into viable application systems. However, the characteristics of the components depend on their size, complexity, and functional capability. Most approaches attempt use of components similar to the common data structure with algorithms as their manipulation. Other approaches attempt use of components resembling functionally complete systems or subsystems that are user interface management systems. Many ways probably exist for use of reusable software components in evolving software systems. However, cited studies suggest that their initial use during architectural or component design specification is a way that implementation can be made quicker. They may also be used for prototyping purposes if a suitable software prototyping technology is available.

Spiral Model

The spiral model involves multiple iteration through cycles with the intent of analyzing the results of prior phases by determining risk estimates for future phases (Figure 18–5). B. Boehm developed the model at TRW in 1988. At each phase, developers evaluate alternatives with respect to the objectives and constraints that form the basis for the next cycle of the spiral. The developer completes each cycle with a review that involves parties with a vested interest. Boehm states, The model reflects the underlying concept that each cycle involves a progression that addresses the same sequence of steps for each portion of the product and for each of its levels of elaboration from an overall concept–of–operation document down to the coding of each individual program.

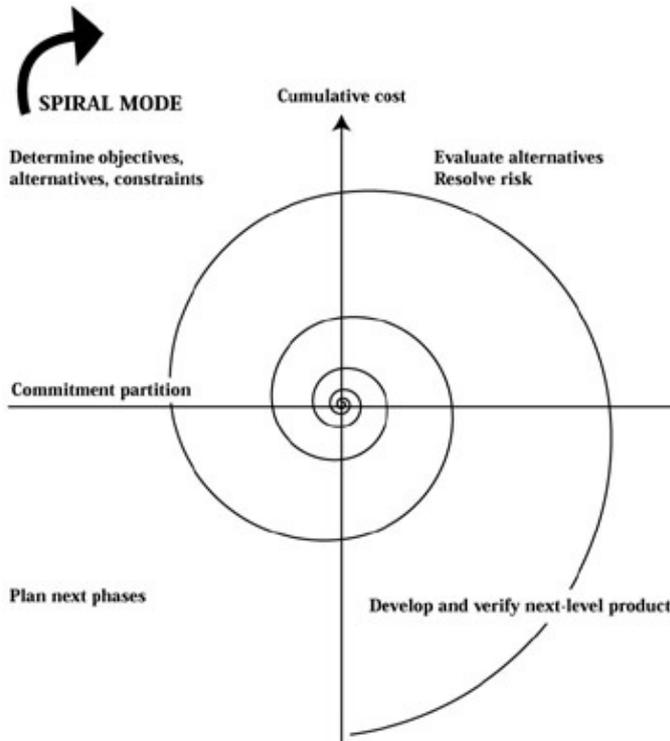


Figure 18–5: A sample of the spiral life–cycle model

The spiral life–cycle model is also a variation of the classic waterfall model. Each of the development phases is carried out in one or more cycles. Each cycle begins with a risk assessment and includes a prototype to provide insight into the risk issues.

The radial dimension represents the added incremental cost incurred in completing the developmental steps. The angular dimension represents the progress made in completing each cycle of the spiral. The basic premise of the model is that a certain sequence of steps is repeated while developing or maintaining software. The steps are first done at a high level of abstraction, and then each loop of the spiral represents a repeat of the steps at successively lower levels of abstraction.

The strength of the model lies in its flexibility for managing a system software life cycle. The developer can plan an examination of risk at each major abstraction. The model accommodates a mixture of specification–oriented, process–oriented, object–oriented, or other approaches to software development. This model is favorable by object–oriented software developers because of its strength as defined previously. A weakness of the model is a lack of matching with any existing standards. The model depends on identification and management of sources at project risk. The model needs more uniformity and consistency. This model assists in the systematic reuse process.

Operational Model

Behaviors particular to the problem domain are modeled and simulated in the beginning stages of the operational model. This is done so that the software customer can explore the way and order in which events happen. Exploration is made possible with the construction of an operational specification of the system. Concern at the specification level with how the system behaves is in contrast to other models whose specifications define the system in terms of a black box that maps the inputs to outputs.

The behavior of the problem domain is emulated in the specification, and the software structures will eventually be used in the actual system, which produce this behavior and are determined later in the

development process.

Transformational Model

The transformational model starts with a program specification and ends with a program. The transformational models progress between the two points is made through an automated series of transformations. H. Partsch and R. Steinbrueggen stated that transformation rules are partial mappings from one program scheme to another so that an element of the domain and its image under the mapping constitute a correct transformation. Transformational programming is a software process of program construction in which successive applications of transformation rule. Usually this process starts with a formal specification of a formal statement with a problem or its solution and ends with an executable program.

The benefit that is associated with the transformational model is the reduction of labor intensity of software production through the automated transformation. This model assists in preserving correctness through the application of formal transformations and replaces the final product testing by verification of the program specifications. One of this models abilities produces a desired transformation through a combination of small units from specialized programming knowledge.

Star Process Model

The star process model is a nonphased model that is useful for user interface activities. User interface development occurs in bottom-up and top-down activities. The star process model is based on empirical evidence from observations of software development. The star process model allows small localized minicycles that are important to user interface development.

Domain Prototype Model

The domain prototype model (DPM) captures in detail the domain-specific knowledge of the application in a form that lends itself to careful point-by-point verification by the domain engineer. The DPM provides, through formal models of the problem domain, a detailed and documented foundation on which requirement decisions are made. The DPM transfers the domain knowledge accurately to the software engineers and communicates the requirements analysis in a form that is easily understood and mapped into design.

DPM concepts do the following:

- Understand the customers requirements
- Show the logic to the customers graphically
- Understand the stated requirements within the domain boundary and environment
- Determine objects
- Establish relationships between objects within the domain infrastructure
- Determine instantiation criteria for objects and their relationships
- Develop functional processes (Figure 18-6)

The domain engineer receives requirements, analyzes, and looks around by accessing domains to see what assets he or she can find to reuse them within the framework.

Domain Functional Model

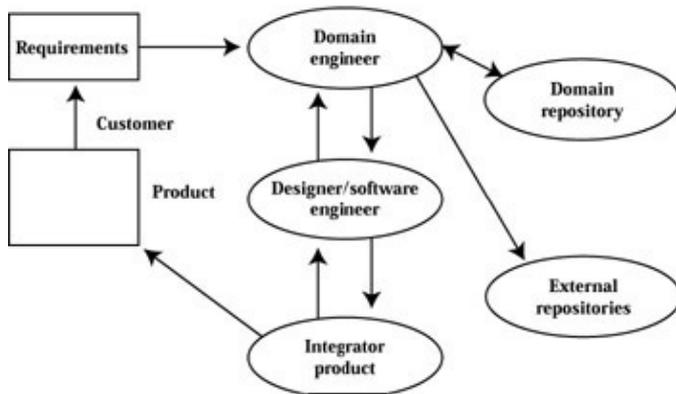


Figure 18-6: A sample of the domain prototype model

Domain Functional Model

The domain functional model (DFM) makes use of data flow diagrams and develops the required processes that drive the objects through their event chains. The following is a suggestive list of processes for the development of a data flow diagram of a single object:

- Develop an object behavior model that formalizes the behavior of an object over time.
- Analyze the action that is performed when the state is entered.
- Break the action down into a sequence of processes.
- Depict each process as a data flow diagram.
- Place each action data flow diagram on a separate page.

Object data are represented as a data store in the data flow diagram. The data store is recorded in the domain data dictionary (DDD). The data store for the object contains all of the attributes of the object. The instances of the object are created and stored in the data store. The data store holds all data and all attributes of all instances of the object. The data store is shared by all action data flow diagrams. The data flow diagram involves data stores that are made up of attributes of that data. These processes are discrete. Data flow diagrams are the offspring of a context diagram. The context diagram establishes the external interfaces of the system (Figure 18-7), and the data flow diagrams identify the internal interfaces of the system (Figure 18-8).

External Interfaces for Requirements

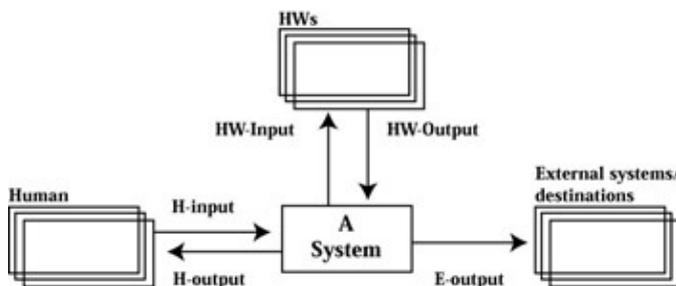
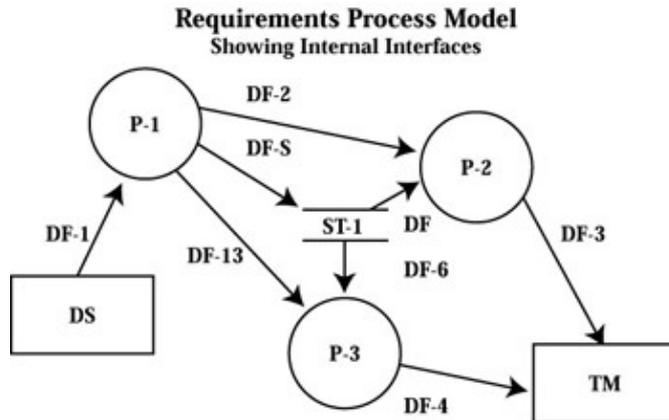


Figure 18-7: A sample context diagram

**NOTATION:**

DF = Data Flow

P = Process (data process)

ST = Store (data store)

TM = Terminator (information destination)

The data flow diagram consists of rectangles that are the source or destination of data outside the system. (Customarily rectangles are not shown in the data flow diagrams, because they are already present in the context diagram.) A data flow symbol (arrow) represents a path where data moves into, around, and out of the system. A process symbol (circle) represents a function of the system that logically transforms data. A data store is a symbol (open-ended rectangle) which shows a place in the system where data is stored.

Figure 18–8: A sample data flow diagram. That data flow diagram consists of rectangles that are the source or destination of data outside the system. (Customarily, rectangles are not shown in the data flow diagrams because they are already present in the context diagram.) A data flow symbol (*arrow*) represents a path where data moves into, around, and out of the system. A process symbol (*circle*) represents a function of the system that logically transforms data. A data store is a symbol (*open-ended rectangle*) that shows a place in the system where data is stored.

DF, Data flow; *P*, process (data process); *ST*, store (data store); *DS*, data source; *TM*, terminator (information destination).

Case Study 18–1

You are proud to have a checking account in The Universe Bank. You are free to write as many checks as you want, as long as you have enough money to cover your checks written in the account. The bank has a good reputation, and there is a \$25 charge for every check that is insufficient. They do not want to keep customers whose checks bounce regularly.

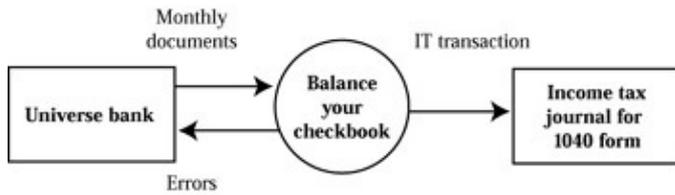
Once a month, the bank's bookkeeper sends your canceled checks, a computerized financial statement, a balance sheet, and other documents, which they process on the bank computer Data Universe. You reconcile your checkbook for errors made by the bank or by you (e.g., while entering the transaction in your checkbook). You also record annual tax-deductible expenses in your yearly journal that will be recorded on a 1040 form. You automate your system that balances your checkbook in the most economical way.

You want to identify reusable components in the requirement. You look around and explore the organization domain repository for reusable assets with similar requirements. The name looks familiar. You access other repositories, including the common object request broker architecture (CORBA), to see if you can reuse available assets.

Software Requirement Specification

Almost all people maintain a checkbook, so a reusable asset must be available somewhere in a repository. The following figures provide a sample solution:

- Figure 18–9 provides a context diagram and identifies external interfaces.



For domain data dictionary:
monthly documents consists of
[cancelled checks deposits ...]

Figure 18–9: A context diagram

- Figure 18–10 partitions the requirements and identifies internal interfaces.

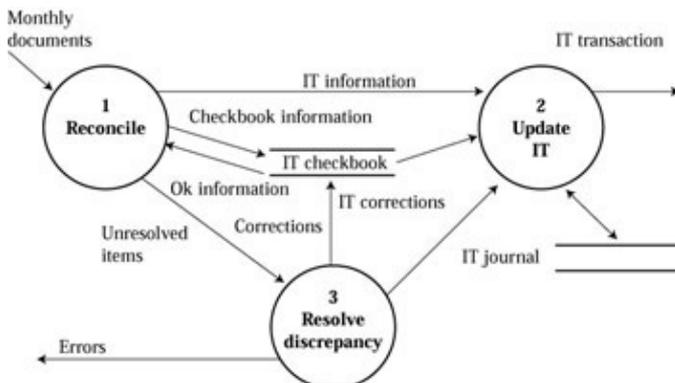


Figure 18–10: A sample of level 1 partitioning

- Figure 18–11 shows that the model can be easily partitioned to create reusable assets in future systems.

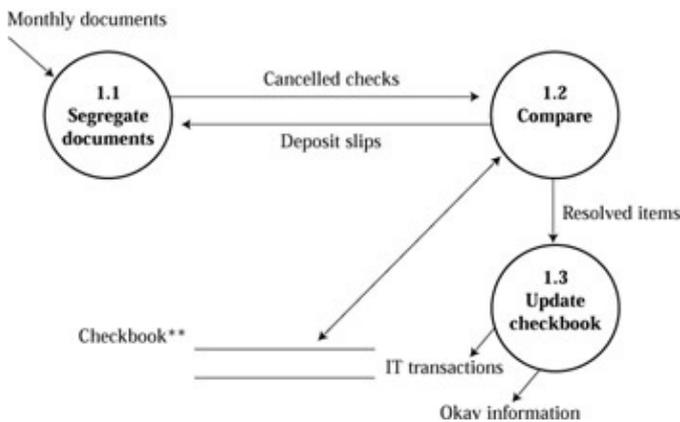


Figure 18–11: A sample of level 2 partitioning

Software Requirement Specification

The software requirement specification contains various activities and tasks for each software item. The IT project manager generates this document to pass on to the software designer, implementers, testers, and

Software Requirement Specification

maintainers. The software requirement specification document includes the following:

- Functional and capability specifications, including performance, physical characteristics, and environmental conditions under which the software item is to perform
- Interfaces that are external and internal to software items
- Verification and validation of software requirements
- Safety specifications
- Security specifications
- Human factor engineering
- Data definition
- Database requirements
- Installation and acceptance requirements for the delivered software
- Quality characteristics
- Configuration management criteria
- User documentation
- User operation and execution requirements
- Training outlines
- User maintenance requirements
- Requirement traceability to system requirements and system design
- Software testing
- Acceptance tests

Chapter 19: Software Design Process

The software design process translates the requirement specification what into another phase of the software development life cycle: how to develop the software. The software design is a blueprint and road map of software development. The software design process allocates the functional requirements to a design structure. This chapter discusses the software design architecture; software design methods, including the object-oriented design method and the structured design methods; and software design descriptions.

Software Design Architecture

The software design architecture represents the physical design of a software system. The architecture primarily captures static information. A model contains one or more architecture diagrams. Multiple architecture diagrams group meaningful collections of objects. Each architecture diagram contains symbols that denote components and dependencies. The components represent structural elements that are provided by the underlying implementation language. Dependencies represent compilation dependencies among components. The symbols for an architecture diagram are somewhat language specific because not all languages provide the same packaging mechanisms, generic units, and tasks.

An architecture diagram takes the form of a graph in which components are vertices of the graph, dependencies are directed arcs, and the topology of the graph satisfies the rules of separate compilation for the programming language.

For each component, a name, its semantics, and relevant design notes are provided. Recording the kind of components that use the vocabulary of the underlying implementation language captures static design decisions. Each component references a set of classes from class structures of the same project model or objects from object diagrams of the same project model. This set of entities registers the design decisions with regard to packaging of the implementation of individual classes and objects. Typically, a one-to-one mapping of classes to components and many-to-one mappings of objects to components exist.

If the rules of underlying implementation language allow, some components explode and reveal the unveiling of another complete architecture diagram. Dependencies denote asymmetric relationships among components.

Production of a diagram motivates changes in corresponding class structures and object diagrams because the logical design is modeled to the physical constraints of the system. These diagrams facilitate the representation and sharing of components among programs, even when such programs execute in a distributed environment.

What is Software Design?

The software design process converts the software requirements analysis into a logical design for the computer software configuration item (CSCI) (Figure 19-1). The software developers further decompose and allocate the requirements to the proper functionality associated components. They construct a hierarchy chart to represent the partitioning of complex functions into simpler functions. They refine software components into lower levels, which contain software units that can be coded, compiled, and tested. The software developers ensure that all of the requirements are differentiated properly and allocated from software components to software units.

Software Design Methods

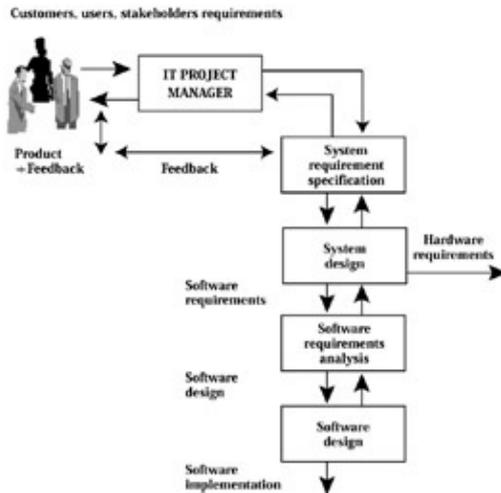


Figure 19–1: IT software design process

The software developers develop a detailed design for the interface external to the software item, between software components, and between the software units. The software requirements are traceable forward and backward. The software developers also develop and document a detailed design for the database.

The software developers define and document test requirements and a schedule for testing software units. The test requirements include stressing the software unit at the limits of its requirements. The software developers update the test requirements and the schedule for software integration. The software developers evaluate the software detailed design and test requirements for the following:

- Traceability to the requirements of the software item
- External consistency with architectural design
- Internal consistency between software components and software units
- Appropriateness of design methods and standards used
- Feasibility of testing
- Feasibility of operation and maintenance

Software Design Methods

Software design methods cover a range of abstraction. A design method includes the choice of environments and approaches to be used for software design. The strategy includes techniques that develop software efficiently. The principal software design methods are the object-oriented method and the structured design method.

Object-Oriented Method

The object-oriented method (OOM) is the combination of an object-oriented approach, a data structure approach with entity-relationship modeling, and a functional approach. The goals of the OOM are to capture in detail the domain-specific knowledge of the application in a form that lends itself to careful point-by-point verification by the domain experts.

Through formal models of the problem domain, the OOM provides a detailed and well-documented foundation on which the software developers make requirement decisions. The method transfers the domain knowledge accurately to the software engineers and communicates the requirement analysis in a form that is easily understood and mapped into an object-oriented design.

Object–Oriented Method

OOM concepts understand the customers requirements, graphically show the logic to the customers, understand the stated requirements, determine objects, establish relationships between objects, determine instantiation criteria for objects and their relationships, and develop functional processes.

An object is a mental abstraction of a set of real world things. This method uses concepts from object–oriented and structured approaches, which includes abstract data types, inheritance, and module coupling and cohesion. It uses graphic models with proper documentation, which transfers the requirements from one phase to another for implementation. The OOM couples its analysis into object–oriented design (OOD) and is implemented in any suitable object–oriented programming language. This concept maintains the traceability of the requirements for embedded systems. The method provides the project controls and communication tools for management, quality assurance (QA), and documentation formats. The following are the basic OOM models:

- Object analysis model (OAM)
- Object information model (OIM)
- Object behavior model (OBM)
- Object process model (OPM)

Object Analysis Model

The OAM consists of all of the analysis that is needed so that the developers can understand the customers requirements. It also contains information with regard to all of the identified external interfaces. This step is especially important for embedded systems. The developers link these interfaces with external systems as illustrated in Figure 19–2. The concept of drawing this graphic is so that the customers requirements are understood in an unambiguous method and the customer understands his or her stated requirements. Identification of processes, data storage, and data flows can extend the model.

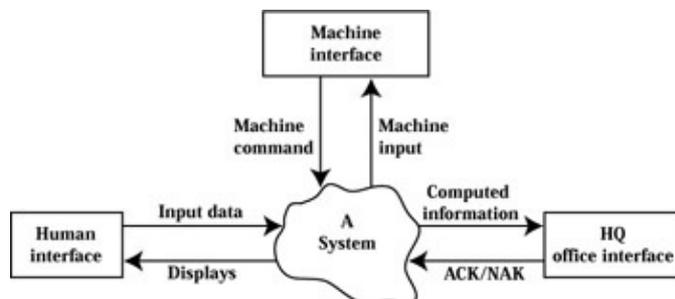


Figure 19–2: An OAM diagram

The goal is that the developers mutually understand and record the requirements in the proper document before proceeding to further analysis or design activity. The developers should clearly define the definition of the notations that are used in the graphic and properly record them in the document for understandability. They can initiate the object data dictionary (ODD) at this level by recording this essential information.

Object Information Model

The OIM identifies the conceptual entities of the problem and formalizes them as objects and attributes. Complete and unambiguous understanding of the problem is the goal. The OIM is the beginning of the design phase. It identifies things about the problem and their relationship. Things and associations are modeled here. The model is simple enough to be easily read and understood. Significant emphasis is placed on formalization of the relationships between objects.

The developers develop a model and depict it graphically as shown in Figure 19–3. They use the textual

Object Attribute

descriptions in the definition of the models semantics. They group like things together into sets (see Figure 19–3). Things are alike if they behave in the same way and can be described by the same characteristics.

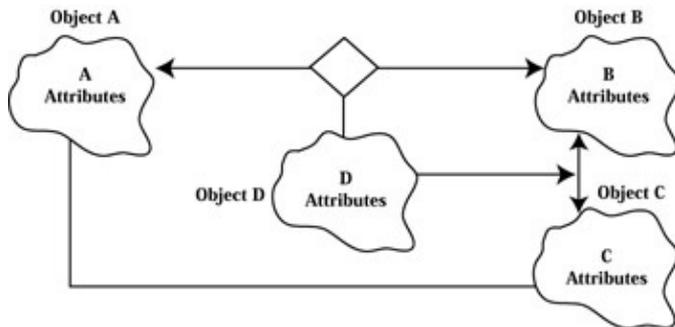


Figure 19–3: An OIM diagram

The characteristics that all elements of the set have in common are called *attributes* of the object. An attribute is the abstraction of a single characteristic, which is possessed by all entities and was abstracted as an object. The set of attributes must be complete, fully factored, and mutually independent.

The following is an example of the dot notation for attribute:

Object Attribute

The types of attributes are descriptive, naming, and referential. The descriptive attributes provide facts that are intrinsic to each instance of the object. The naming attribute provides a fact about the arbitrary name carried by each instance of the object. The referential attribute links an instance of one object to an instance of another object.

The developers use tables to define the types of questions that can be answered by inspection of the objects. The table name is the name of the object. Each column of the table is an attribute of the object, and each row is an instance of the object. The instance is the specific element of the set and is denoted by that object name. As illustrated in Figure 19–4, every box in a table contains exactly one value. Attributes of an object should not contain internal structure. Tables are the basic formal structure of the OIM because they are simple and adequate. The object representation by table assists in the identification of the objects and attributes. The table also helps in the representation of instances of objects.

Object A		
Attribute A1	Attribute A2	Attribute A3
Instance - A		

Figure 19–4: OIM table

Associations exist between things in the real world. These associations must be formalized in this model. These can be recognized by verb phrases in the descriptions. For example:

The car *has* tires.

Object Attribute

Thus a relationship is named by a verb phrase. It can be phrased in both directions of the relation. For example:

A class is composed of students.

or

Students comprise a class.

The same two objects may have more than one relationship between them, depending on their type of relationship. An object may be related to itself. A relationship can also exist between multiple objects. For example:

The floppy disk was formatted on the disk drive.

The floppy disk is owned by a student.

The floppy disk contains disk files.

This type of relationship, which involves two objects, can be classified into three fundamental forms and called *multiplicity*:

Multiplicity	Notation	Notation	Example
One-to-one		1:1	The husband has a wife.
One-to-many		1:M	The student owns books.
Many-to-many		M:M	The capacitor is a component of a computer.

A graphic notation is necessary for the replacement of a table for complex objects (Figure 19-5). The graphic notation eases handling of complex systems and represents spatial compression of model. Figure 19-6 illustrates OIM notation representations. Figure 19-7 shows supertype and subtype constructs. Figure 19-8 illustrates the correlation relationship symbol.

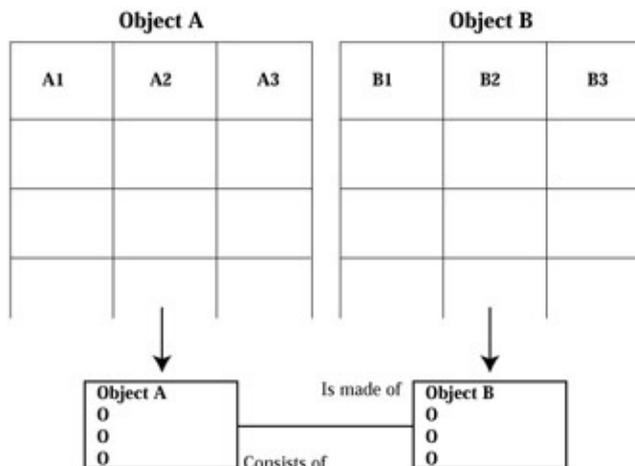


Figure 19-5: OIM table and object relationship

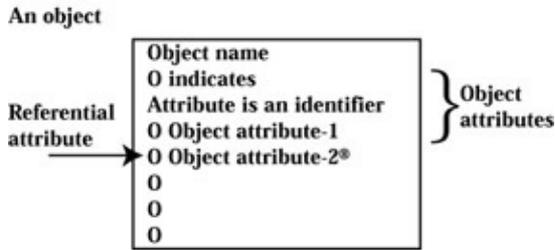


Figure 19-6: OIM notations

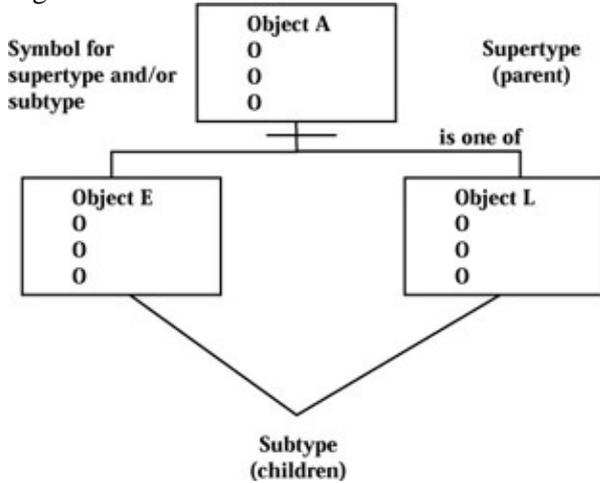


Figure 19-7: Supertype and/or subtype constructs

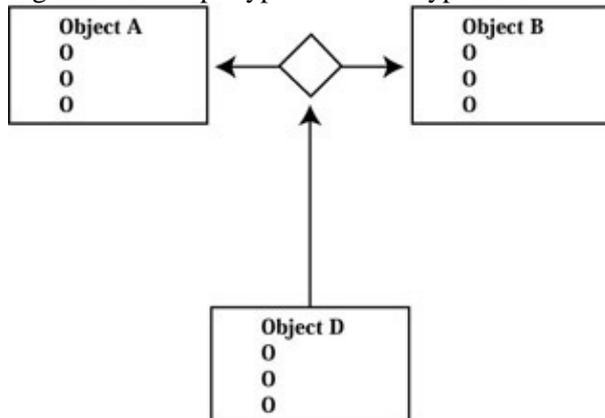


Figure 19-8: Correlation symbol

Object data dictionary. The developers document the object, attributes, and their relationships in an ODD. This document consists of a written description of objects, formalizes the identification of objects, forms part of the formal system specification, and separates descriptions that are written for each object. This is a live document and should start at the beginning of the software engineering phase and be enhanced throughout the software development and maintenance phases.

Selection criteria for a correct object. Four major tests help by not accepting false objects:

1. **The Uniformity Test** is based on the definition of an object, and each instance of the object must have the same set of characteristics and be subject to the same rule.
2. **The More-Than-a-Name Test** is applicable to objects that cannot be described by attributes. This object has no characteristics other than its name and is probably an attribute of another object.
3. **The Or Test** is conducted on the object description. If the inclusion criteria in the object description uses the word or in a significant manner, a set of diverse things is present rather than an object.

Object Behavior Model

4. *The More-than-a-List Test* is conducted on the object description. If the inclusion criteria are in the object description, then it is simply a list of all specific instances of the object and is most likely not a true object.

Object Behavior Model

The OBM formalizes the life or event histories of objects and relationships as was identified in the OIM. Things go through various stages during their lifetime in the real world. The life cycle of an object is therefore the behavior of an object during its lifetime. Figure 19–9 presents an example of the object lifetime diagram.

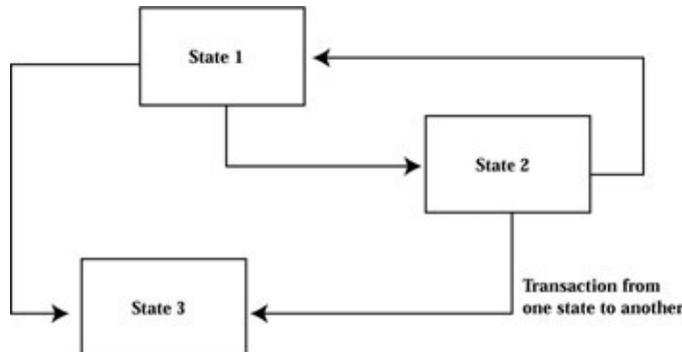


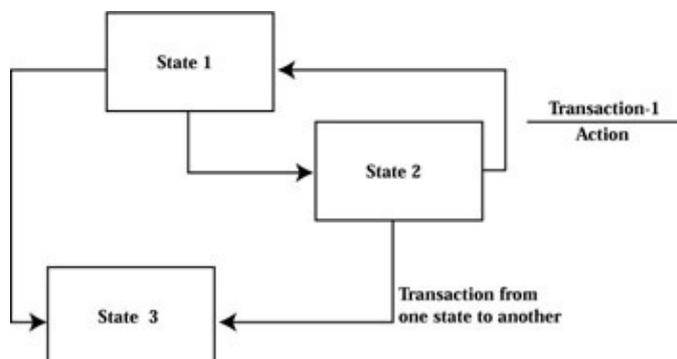
Figure 19–9: Object lifetime diagram

An object may be in only one stage at a time. The stages are mutually exclusive, they are discrete, and transitions can occur instantaneously. Transitions from any stage to any other stage are not always allowed. Incidents cause transition of things between stages. Some incidents cause a progression only when the thing is in certain stages of its life cycle. Similar characteristics of real world things have a common life cycle. Thus the OBM is a formalization of the life cycle of an object in regard to the following parts:

- States
- Events
- Transitions
- Actions

The state of OBM corresponds to the state of an objects life cycle. An *event* is an incident or action that causes a progression to another state or to the same state. The *transition* is the new state of an object if a particular event occurs while the object is in a particular state. The *action* is the function that is performed immediately when entering a new state. Each state can have only one action, but that action can consist of many processes.

Figure 19–10 shows a sample OBM. The boxes represent the states, and the lines represent the events. The event causes the transition to the new state.



Object Behavior Model

Figure 19–10: OBM sample

An alternate form of state model representation is the state transition table (STT). In an STT the rows represent states, the columns represent events, and the cells represent the effect of each event in each state. When a particular event cannot happen for a particular state of the object, then the cell entry cannot happen.

The event may be prevented from happening for many reasons, such as a physical impossibility, definition of the object, or constraints. If a particular event can happen for a specific state of the object but the object does not respond to the event, then an event–ignored entry is made in that cell. If the event is ignored, then the object stays in the same state. The advantages of an STT are that it can be extended by adding an additional column for action and is associated with entering each state. The information contained in the STT or state transition diagram (STD) is the same, but the development of the STT from the STD will catch one of the most common errors in OBM development.

OBM is built for each dynamic object in the OIM. The following are guidelines for construction of an OBM:

- Take one instance of the model and analyze and record the life cycle of that instance.
- Write down the various states for that instance.
- Find the states that consider all relationships of the object.
- Build an STT.
- Check the STT for new or additional states.
- Define the action that will be executed upon entry into a state.
- Identify nonfinal states.
- Identify events that the object must generate so it can exit nonfinal states.
- Expand actions that generate these events.
- Add the new events to the STT.
- Complete the STT.
- Complete the STD.

Object Process Model

The OPM makes use of data flow diagrams and develops the required processes that drive the objects through their event chains. Only a few concerns with objects are discussed in this section.

The following is a list of processes for the development of a data flow diagram of a single object:

- Develop an OBM that formalizes the behavior of an object over time.
- Analyze the action that is performed when the state is entered.
- Break the action down into a sequence of processes.
- Depict each process as a data flow diagram.
- Place each action data flow diagram on a separate page.

The data of an object are represented as a data store in the data flow diagram. The data store will be recorded in the ODD. The data store for the object contains all of the attributes of the object. The instances of the object are created and stored in the data store. The data store holds all data and all attributes of all instances of the object. The data store is shared by all action data flow diagrams. The data flow diagram involves data stores, which are made up of attributes of that data. These processes are discrete.

Structured Design Method

The structured design method maps the flow of data from its problem domain into its software structure (Figure 19–11). The steps of structured design involve characterization of the data flow through graphic representation, identification of the various transform elements, assembly of these elements in a hierarchical program structure, and refinement and optimization of the elements. The structured design uses the following terms:

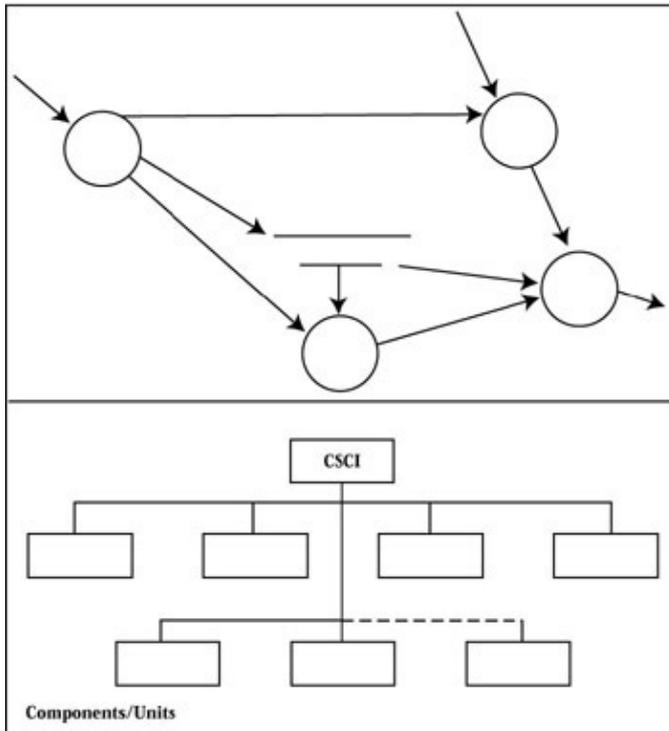


Figure 19–11: Software-structured design

- **Efferent.** This is the flow of information out of a software system, such as the output flow. Efferent modules take information from superordinates, possibly perform some data transformation, and then pass the information onto another subordinate module. The primary purpose of afferent and efferent modules is to pass information into and out of the software system. As afferent modules pass information into the software system, the information becomes less recognizable as input data, which is more abstract. As the efferent flow is traced backward into the software system, it becomes less recognizable as output data. Points of higher abstraction of input and output data are further removed from the physical inputs and outputs. These points still constitute as inputs to and outputs from the software system. The points of the highest abstraction for the major input and output streams define the afferent, efferent, and central transformation portions of a software hierarchy. Structured design does not provide precise guidelines for determining the points of highest abstraction. Designer intuition and experience are required to make these determinations.
- **Central transforms.** These are modules whose primary function is to transform information from one form into another and perform the primary work of the software system. Transform flows move data from the invoking module to the subordinate module, if the subordinate module has performed some transformation, and then back to the invoking module.
- **Structure chart.** Developers use the data flow to develop a structure chart by performing top-down decomposition of the central transforms. The structure charts present the partitioning, hierarchy and organization, and communication among software modules. A module is a set of one or more program statements that is invoked by other parts of the software system. Developers use the structure chart to

Software Design Description

develop the data structure. They use the results to reinterpret the functional requirements as described in the software requirements specifications. A module is equivalent to a software item, unit, or component.

- **Fan-out.** This is the number of subordinate modules that a module possesses. A high fan-out can lead to a flat software system that has no intermediate levels in its structure
- **Fan-in.** This is the number of invoking modules that a module possesses. A fan-in means that duplicated code was avoided.
- **Coupling.** This is a measurement of relationships among modules and is used to evaluate various program organizations. It is a measure of the strength of interconnection when the strength of coupling and intermodule dependence is directly related. Strong coupling complicates a system because a module is hard to understand, change, or correct by itself if it is highly interrelated with other modules. Developers can reduce complexity by designing a module with the weakest coupling possible between modules.
- **Cohesion.** This is a measurement of the strength of association of elements within a module. Modules with a high degree of cohesiveness are understandable and excellent candidates for reuse.
- **Heuristic design.** In addition to coupling and cohesion, structured design discusses a heuristic design that is used to organize modules and decisions. Developers use it as a check or indicator by which a structure can be examined for potential improvement.
- **Module size.** The size of the module should be small enough to be understandable.
- **Span of control.** This is the number of modules immediately subordinate to a given module and should be between 2 and 9. Spans of control beyond this range usually indicate that the modules function is too complex.

When assessing the wide popularity of structured design, L. Peters (1981) stated the following:

Structured design has gained wide popularity for two primary reasons. One is that it allows the software designer to express his perception of the design problem in terms he can identify with: data flows and transformations. The notation with which he expresses these flows is simple, easy to use, and understandable by management, customer, and implementers.

The other primary reason for this methods popularity is that it provides the designer with a means of evaluating his (and others) design. This serves as a benchmark against which to measure his success or progress. The method is unique in this regard. If the design evaluation concepts consisted of nothing more than coupling and cohesion, structured design would still be a significant contribution to the software fields.

Software Design Description

The software design description (SDD) contains a detailed description of a software item. With the software architecture, it includes all the detail that is needed to implement the software. The SDD is a representation of a software system that is used as a medium for communication of software design information. The SDD contains internal interfaces among the software components and units, networking design, and database design. The SDD includes the following:

- Software design architecture
- Description of how the software item satisfies the software requirements, including algorithms and data structure
- Software item input and output description
- Software item relationships
- Concept of execution, including data flow and control flow
- Requirements traceability

Software Design Description

- ◆ Software component–level requirements traceability
- ◆ Software unit–level requirements traceability
- Rationale for software design
- Reuse element identification
- Definition of types of errors and their handling
- **Design entities and attributes.** A design entity is an element (component) of a design that is structurally and functionally distinct from other elements and that is separately named and referenced. Design entities result from a decomposition of the software requirement. A design entity attribute is a named characteristic or property of a design entity. It provides a statement of fact about the entity. An incorrect specification of the attribute value can result in a fault in the software implementation.
- **Identification.** Unique name of the design entity. Two entities should not have the same name.
- **Type.** Description of the kind of entity, such as subprogram, module, procedure, process, or data store. Alternatively, design entities may be grouped into major classes to assist in locating an entity dealing with a particular type of information.
- **Purpose.** Description of why the entity exists (rationale for the creation of the entity). The purpose attribute should also describe special requirements that must be met by the entity that are not included in the software requirement specification.
- **Function.** A statement of what the entity does. The function attribute states the transformation applied by the entity to input to produce the desired output. In the case of a data entity, this attribute states the type of information stored or transmitted by the entity.
- **Subordinates.** Identification of all entities composing this entity. This information is used to trace requirements to design entities and to identify parent–child structural relationships through a software requirement decomposition.
- **Dependencies.** A description of the relationships of this entity with other entities. These relationships are graphically depicted by structure charts, data flow diagrams, and transaction diagrams. This attribute should describe the nature of each interaction, including such characteristics as timing and conditions for interaction. The interactions involve the initiation, order of execution, data sharing, creation, duplication, usage, storage, or destruction of entities.
- **Interface.** A description of how other entities interact with this entity. The interface attribute describes the methods of interaction and the rules governing those interactions. The methods of interaction include the mechanisms for invocation or interruption of the entity, communication through parameters, common data areas or messages, and direct access to internal data. The rules governing the interaction include the communications protocol, data format, acceptable values, and the meaning of each value. The interface attribute provides a description of the input ranges, the meaning of inputs and outputs, the type and format of each input or output, and output error codes. It also includes inputs, screen formats, and a complete description of the interactive language.
- **Resources.** A description of the elements used by the entities that are external to the design. The interaction rules and methods for using the resource are specified by this attribute. The attribute provides information about items such as physical devices (printers, disc–partitions, memory banks), software services (mathematics libraries, operating system services), and processing resources (central processing unit cycles, memory allocation, and buffers). The resources attribute describes usage characteristics, such as the process time at which resources are to be acquired and sizing to include quantity, and physical sizes of buffer usage. It also includes the identification of potential race and deadlock conditions as well as resource management facilities.
- **Processing.** A description of the rules that the entity uses to achieve its function. The processing attribute describes the algorithm that the entity uses to perform a specific task and includes contingencies. The description includes timing, sequencing of events or processes, prerequisites for process initiation, priority of events, processing level, actual process steps, path conditions, and loop back or loop termination criteria. The handling of contingencies describes the action to be taken in the case of overflow conditions or in the case of a validation check failure.

Software Design Description

- **Data.** A description of data elements internal to the entity. The data attribute describes the method of representation, initial values, use, semantics, format, and acceptable values of internal data. The description of data is in the form of a data dictionary that describes the content, structure, and use of all data elements. It includes data specifications such as formats, number of elements, and initial values. It also includes the file structures, arrays, stacks, queues, and memory partitions. The description includes static versus dynamic data elements; whether it is to be shared by transactions, used as a control parameter, or used as a value; and loop iteration count, pointer, or link field. The data information includes a description of data validation needed for the process.

Chapter 20: Software Implementation Process

Overview

This chapter discusses software implementation, software implementation with reuse, software testing, and configuration management (CM). The software implementation process includes the following:

- Writing source code, execute code, and test code for each identified item in the design phase
- Integrating software units and software components into software items
- Conducting software units and software components testing to ensure that they satisfy the requirements

Software Implementation

Software implementation covers coding of decomposed computer software configuration items (CSCIs) with the selected computer language. Each coded software units and components, as shown in Figure 20–1, are tested and documented in the software development files (SDF). Testing is no longer an activity that starts only after the coding phase is complete with the limited purpose of finding mistakes. Software testing is an activity that encompasses the whole development process and is an important part of the implementation.

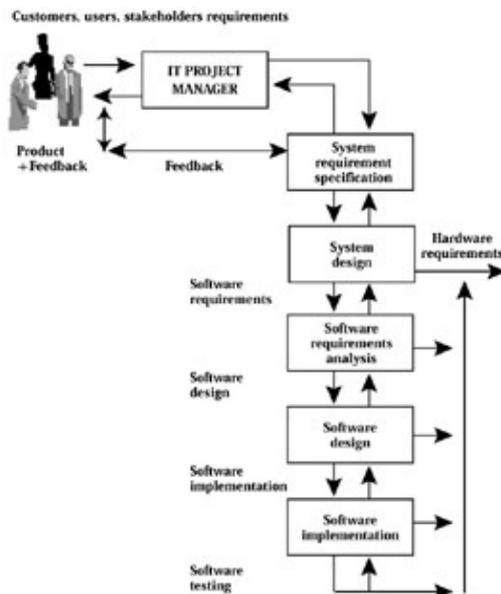


Figure 20–1: IT software implementation process

The IT manager begins to plan for testing at early stages of requirements analysis and design. He or she systematically and continuously refines the test plans and procedures as the development proceeds. These activities of planning and designing tests constitute a useful input to designers for highlighting potential weaknesses. Testing is seen as a means primarily for demonstrating that the prevention has been effective and for identifying anomalies in those cases in which it has not.

Software Implementation with Reuse

Software implementation with reuse is reality. The benefits of reusable software components are achieved if the reusable software is successfully used in another application development within the domain or in any

New Development

other domain. Reusability is one of the promises of the object-oriented approach, but the user must plan in advance to achieve the benefits of reuse. Many different versions of reuse exist in the industry, including code, inheritance, template, component, framework, pattern, artifact, and asset. The bottom line is not reusable; it cannot be used more than once.

Identification of software depends on whether it is a new development, reuse of the available existing software, reengineering of the software, or a reverse engineering process. Many commercial-off-the-shelf (COTS) software products are available that suit the requirements. Domain knowledge and experience of the system analyst or engineer play a major role in the selection decision. The selection must be based on development of the right software for the right requirements.

New Development

In new development, various software development phases are met. The major phases are domain analysis, requirements analysis, design, and implementation. Developers identify commonalities and differences in domain analysis. These can be further divided into subphases, each of which are reiterative. This is an example of a typical waterfall model. The models must be shared among the group of developers who seek to integrate software. The model must be well designed, consistent, and based on a proper design method. The model may be constantly evolving with new requirements or modifications.

The new development normally follows a set of prescribed standards, methods, and computer-aided software engineering (CASE) tools. Traditionally, the process moves from high-level abstractions and logical implementation-independent designs to the physical implementation of a system. It follows a sequence from requirements through the design of the implementation. It leads forward to a new development of software throughout the life-cycle phases. The process starts at the initial phase of analysis of the new requirements and progresses to the development of all phases of analysis until the project is completed.

Reusable software is developed in response to the requirements for one application and can be used in whole or in part for the satisfaction of requirements of another application. The domain analyst must find a way for the existing software that is pretested and cost and time efficient to be reused. The basic approach is configuration and specialization of preexisting software components into viable application systems. Cited studies suggest that initial use of reusable software during the architectural design specification is a way to speed up implementation.

Reverse Engineering

Reverse engineering extracts design artifacts and the building or synthesizing of abstractions that are less implementation dependent. This process implements changes that are made in later phases of the existing software and automatically brings back the early phases. It starts from any level of abstraction or at any stage of the life cycle. It covers a broad range that starts from the existing implementation, recaptures or recreates the design, and finally deciphers the requirements that the system implements.

Reengineering

Reengineering is the renovation, reclamation, examination, and alteration of the existing system software for changing requirements. This process reconstitutes the existing system software into a new form and the subsequent implementation of the new form. This dominates during the software maintenance life cycle. This helps the user identify and separate those systems that are worth maintaining from those that should be replaced.

Exploration of the Internet for Suitable Reusable Assets

Exploration of the Internet for suitable reusable assets is a mechanism to integrate reuse into software implementation. This is a process of building software systems by integration of reuse assets from the Internet repositories. Figure 20–2 shows a scheme for integration of software systems with reuse assets. For example, the software development team receives and analyzes new requirements for a system. The domain engineer also examines the requirements and confirms whether the requirement is already present in any domain repository. The software developers get the reusable well–tested assets and save time and money in the integration of a software system. Sometimes the developers may have to write and test software interfaces.

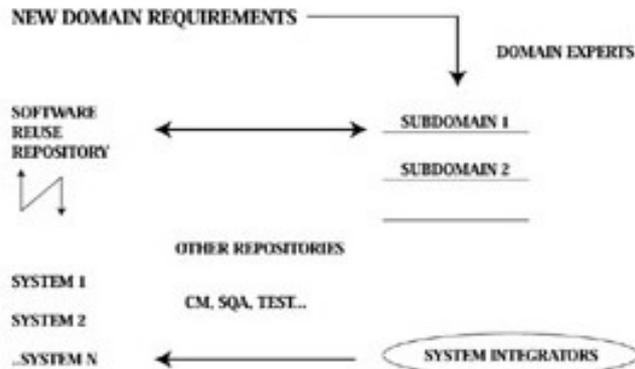


Figure 20–2: Integration of software systems with reuse assets

Case Study 20–1

The town department of Fort Cobb, Oklahoma, needs a traffic light (TL) on the main street. About 3000 residents live in this town, most of whom travel for jobs to nearby cities. Another road intersects the main street.

The object TL establishes a proper relationship with other objects, such as a clock, traffic sensor, traffic, intersection, and control panel. The TL attributes are manufacturer, model, serial number, and road intersection name. The clock attributes are manufacturer, model, serial number, TL manufacturer, TL model, and TL serial number. The traffic sensor attributes are manufacturer, model, serial number, TL manufacturer, TL model, TL serial number, and road name. The traffic attributes are road name, TL manufacturer, TL model, and TL serial number. The intersection attribute is road name. The control panel attributes are manufacturer, model, serial number, button selected, TL manufacturer, TL model, and TL serial number.

This case study and others discussed in this chapter are common scenarios in the industry. An important question is why these scenarios have to be designed from the beginning when enough information is available for reuse of assets and savings of time and money. Figure 20–3 shows a sample partial solution.

Software Testing

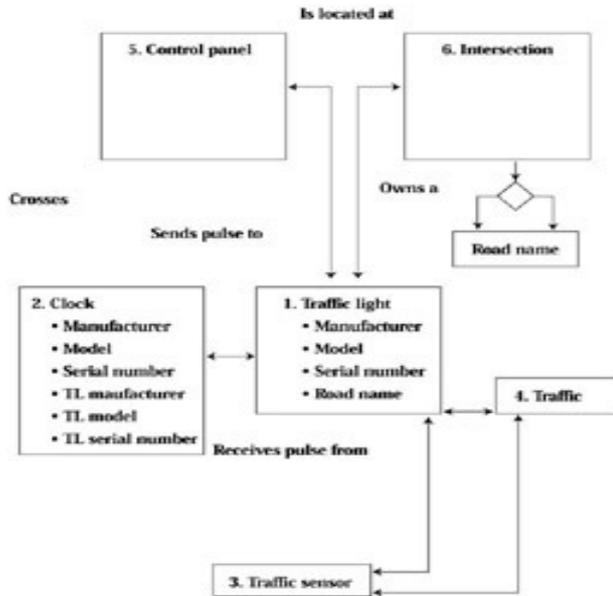


Figure 20–3: A sample solutions

The software implementers test each software unit and database, ensuring that it satisfies its requirements. They document the test procedures, data, and test results for each software unit and database in an SDF. They evaluate the software code and test results for the following criteria:

- Traceability to the requirements and design of the software item
- External consistency with the requirements and design of the software item
- Internal consistency between unit requirements
- Test coverage of units
- Appropriateness of coding methods and standards used
- Feasibility of software integration and testing
- Feasibility of operation and maintenance

The software implementers update the user documentation as necessary. They also update the test requirements and schedule for software integration.

Software Testing

Software testing is the verification and validation of the specific requirement. Verification means that software analysts check the observed behavior against functional specification. Validation means that analysts check the observed behavior against the customers expectations. The software testing consists of the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite execution domain, against the specified expected behavior. Dynamic testing means the code execution and implies execution of the program on valued inputs. Static analysis techniques involve peer review and inspection program execution on symbolic inputs or symbolic evaluation. Static testing indicates no code execution.

Many test cases are theoretically possible for even simple programs in which exhaustive testing could require years to execute. The number of executions that can realistically be observed in testing must be manageable. Testing always implies a trade–off between limited resources, schedules, and inherently unlimited test requirements. The IT manager decides whether the observed outcomes of program execution are acceptable; otherwise the testing effort would be useless. Software testing is a means for quality evaluation.

Software Testing Best Practices

Software testing best practices includes static and dynamic analysis techniques as discussed in the following sections.

Static Analysis Techniques

- Review, walk through, and inspection practices use visual review of a portion of technical documentation to detect errors. They are usually conducted by small peer groups of technical professionals who have vested interest, using software requirement documents, specification, design description, and source code listings to do a line-by-line inspection, walk-through, and reviews. These practices ensure that the differentiated software requirements are present and satisfied.
- The code auditor is a software programmer who examines the source code listing that he or she follows and complies with set standards and practices.
- The interface checker uses automated tools to analyze software requirements specification, design description, and source code listing to detect errors in information control passed between software units and components.
- Physical units. An automated tool to ensure consistency in software units that are involved in computations conducts testing. These practices ensure that the resultant units are rational in the real world.
- Data flow analysis practices ensure that sequential events occur as scheduled.
- Structural analysis detects violations of control flow standards, including improper calls to routines and subroutines, infinite loops, and recursive relationships in the source code listing.
- Cross-reference programs compile a list of where each data item is used in the source code listing.
- Path analysis generates test data sets to evaluate selected paths in the source code.
- Domain testing detects logical errors when input follows the wrong path.
- Partition analysis detects missing paths, incorrect operators, domain logical errors, and computation errors.
- Complexity analysis examines coded algorithms for simplicity.

Dynamic Analysis Techniques

- Cause-effect analysis uses input data systematically selected to have a high probability of detecting errors in output data.
- Performance-measuring techniques monitor software execution to detect code or execution inefficiencies for evaluation of central processing unit (CPU) cycles and waiting times.
- Path and structural analysis monitors the number of times that a specific line of source code is executed. The source code is classified as statements, branches, or a path.
- Interactive debugging allows the programmer to suspend execution at any point and examine the program status.
- Random testing uses random samples of input data and evaluates the resulting output. These practices generate unexpected conditions to evaluate program performance in those cases.
- Function testing is the most commonly practiced technique. This executes source code using specified controlled input to verify that the functions are performed as expected.
- Mutation analysis studies the behavior of a large number of different versions of the original program. Introducing a small number of errors in each version and studying how the software system responds can generate these practices.
- Error seeding is a statistical approach used to determine the number of errors remaining in large software systems by determining the effectiveness of testing. These practices determine the number of known errors that are detected, indicating the number of unknown errors remaining.

Case Study 20-2

A message processing system is to be developed to provide user communication with a central host computer. The user can compose a message, print a message on a local printer, send a message to a host, read a message from a host, and log a message sent to a host in a local floppy disk file.

A message is a free-format group of characters up to 1024 bytes long. A message buffer is circular, and if the message length exceeds 1024 bytes, then the first character will be overwritten. Commands from the user are single keystroke control codes:

- CTRL-P prints the message.
- CTRL-S sends the message.
- CTRL-R reads the message received from the host.
- CTRL-L invokes the logging control submenu.

As messages are received from the host, they are stored in a buffer. Each message received overwrites the message previously stored in the buffer so that the user can read only the latest message received at any particular time. The user can read the messages as many times as he or she wishes. If the user is reading a host message, then he or she must press the escape (ESC) key to resume message composition. He or she can log messages that are sent to a local floppy disk. The user controls this through a submenu activated by the CTRL-L command.

This case study is a typical example of a real-time system. Figs. 20-4 to 20-6 show a sample solution. In Figure 20-4, a dotted line indicates one bit of information and is represented by the label CTRL-L. In Figure 20-5, the storage buffer is shown a few times to avoid confusion. Each duplicate buffer is noted by *. The dotted circle number 5 will be further transformed into STD as shown in Figure 20-6.

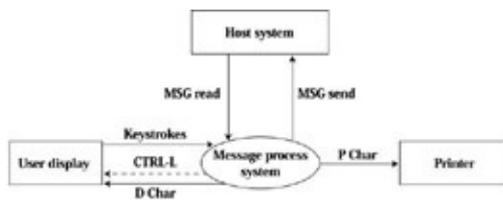


Figure 20-4: Functional model (context diagram)

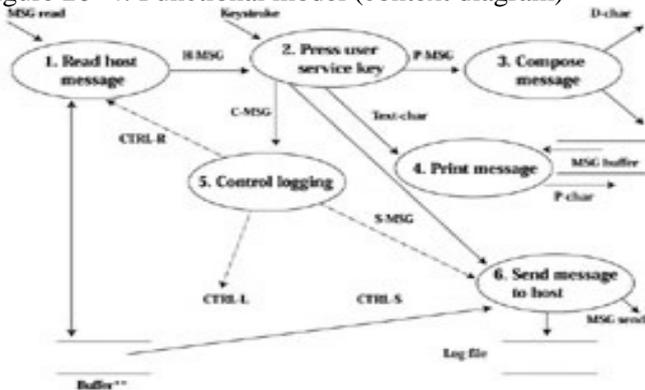


Figure 20-5: Functional model (data flow diagram)

Software Integration Testing

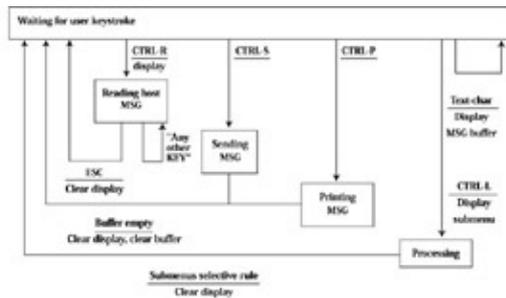


Figure 20–6: STD sample solution

As the software developer writes source codes for a requirement or modifies an existing software unit, he or she formulates and develops software test cases and procedures to verify that the completed software operates correctly. Once the developer tests and verifies the software unit, he or she integrates it with other software system units. Together they are logically cohesive to the modified software units, thus creating software components.

At the software component level, the software developer again conducts a series of tests to verify that the modified software operates correctly and interfaces with other software components that are associated with the modified software unit. The software developer determines how comprehensive the testing must be to verify that the modified software functions correctly and interacts properly with other software units that use the output from the modified software unit.

At a minimum, the software developer must conduct tests to verify the correct operation of all units that are affected by the modified software units. If a software unit is dependent on another software unit that has been modified, then the developer must also test the dependent software unit. The software developer also conducts tests on other integrated software units that are affected by the modified software unit. Once the developer verifies that the software components operate correctly, he or she integrates it in an iterative process with the remaining modified software components until the entire CSCI is built or rebuilt.

Software Integration Testing

Software integration testing is a plan to integrate software units and software components into the software item for testing. The IT project manager establishes the plan and includes test requirements, procedures, data, responsibilities, and schedule. The software implementers integrate the software units and software components and test as the aggregates are developed in accordance with the integration plan. The project manager ensures that each aggregate satisfies the requirements of the software item and that the software item is integrated at the conclusion of the integration activity.

For each qualification requirement of the software item, the software implementers develop and document a set of tests, test cases, use cases (inputs, outputs, and test criteria), and test procedures for conducting software qualification testing. The software implementers evaluate the integration plan, design, code, tests, test results, and user documentation considering the following criteria:

- Traceability to the software system requirements
- External consistency with the software system requirements
- Internal consistency
- Test coverage of the requirements of the software item
- Appropriateness of test standards and methods used

Configuration Management

- Conformance to expected results
- Feasibility of software qualification testing
- Feasibility of operation and maintenance

The software implementers conduct the software qualification test (SQT). Upon successful completion of the SQT, the software developers establish a baseline for the design and code of the software item. They place the software unit components under CM control.

Configuration Management

CM is a discipline that applies technical and administrative direction and surveillance to identification and documentation of the functional and physical characteristics of a configuration item. The CM activities are grouped into four functions:

1. **Configuration identification.** Configuration identification activities identify, name, and describe the documented physical and functional characteristics of the code, specifications, design, and data elements to be controlled for the project. Controlled items are the intermediate and final outputs, such as executable code, source code, user documentation, program listings, databases, test cases, test plans, specifications, and management plans. Controlled items include the support environment, such as compilers, operating systems, programming tools, and test beds. The software implementers control changes to those characteristics and record and report changes to processing and implementation status. The configuration item (CI) is an aggregation of hardware and computer programs or any discrete portion thereof that satisfies an end–use function and is designated by the customer or software system developers for CM.
2. **Configuration control.** Configuration control activities request, evaluate, approve or disapprove, and implement changes to baseline CIs. Changes encompass error correction and enhancement.
3. **Status accounting.** Configuration status accounting activities record and report the status of the project CI.
4. **Configuration audits and reviews.** Configuration audits determine to what extent the CI reflects the required physical and functional characteristics. Configuration reviews are management tools for establishment of a baseline.

The IT manager establishes the CM section. The objective of CM is to assist the IT manager in achieving the following:

- Lowest life–cycle cost for software development and maintenance
- Good performance
- Realistic schedule
- Operational efficiency
- Logistic support
- CI readiness
- Audit trail

The IT manager introduces appropriate CM controls throughout the softwares development and maintenance phases. This policy achieves maximum efficiency in management of changes with respect to the cost, timing, and implementation, and it ensures uniformity in policy, procedures, data, forms, and reporting throughout the organization.

The IT manager should clearly define the requirements for a CM plan at the early stages of the software systems development and maintenance. The manager should employ baselines throughout the life cycle of a

Software Change Process

CI to ensure an orderly transition from one major commitment point to the next in the software systems development, production, and logistic support processes. The manager should establish baselines at those points in a program where it is necessary to define a formal departure point for control of future changes in performance, design, production, and related technical requirements.

Software Change Process

A small change in software can start a chain reaction. The software change process helps in the management of such a chain reaction. The software developers should limit changes to those that are necessary or offer significant benefits. The degree of benefits is directly related to the promptness with which actions are processed. The software developers should make changes only for the following reasons:

- To correct deficiencies
- To satisfy changes in operational or logistic support requirements
- To effect substantial life-cycle cost savings
- To prevent or allow desired slippage in an approved schedule

Preparation of the Engineering Change Proposal

The engineering change proposal (ECP) requires a complete analysis of the influence of the implementation of the software change that it proposes. It requires that the proposal submitted with an ECP contain a description of all known interface effects and information concerning changes required in the functional/allocated/product baselines. The ECP further requires the submission of supporting data outlining the influence on integrated logistic support and overall estimated cost.

Configuration Control Board

The configuration control board (CCB) decides to take proper action concerning software change evaluations, discrepancy reports, processing, approval or disapproval, and implementation. The most important activity of the CCB is the evaluation of discrepancy reports and requests for software changes. The following are some of the factors that can guide decision making:

- Size of the change
- Time
- Complexity of the change
- Influence on the system architecture
- Cost
- Criticality of the area involved
- Influence on other software changes in progress
- Test requirements
- Available resources

The CCB is the official agency to act on all proposed software changes. The chairman of the CCB makes the final decision on all changes unless otherwise stated. The CCB critically evaluates every proposed configuration change, taking into consideration all aspects of the change on a CI and the associated CIs with which it interfaces. Such aspects include the following:

- System design
- Software design

Software Change Process

- Reliability
- Performance reliability
- Maintainability
- Cost
- Schedule
- Operational effectiveness
- Safety
- Security
- Human factors
- Logistic support
- Transportability
- Training
- Reusability
- Modularity

Section V: IT Project Completion and Evaluation

Chapter List

Chapter 21: System Integration and Evaluation

Chapter 22: Practical Case Study

A good management practice is to successfully complete an IT project within budget and schedule.

Chapter 21: System Integration and Evaluation

This chapter covers the system integration process, system requirements verification and validation (V & V), documentation, customer acceptance testing, future trends, and guideline suggestions for a successful IT system project.

System Integration Process

The system integration process integrates hardware configuration items with software configuration items, manual operations, and other systems as necessary, into the IT system (Figure 21–1). The system developers test aggregates against their requirements as they are developed. For each requirement of the system, the system developers create a set of test cases and test procedures for conducting the system integration test. The test cases include inputs, outputs, test criteria, and use cases. The system developers evaluate the integrated system as follows:

- Coverage of system requirements
- Appropriateness of test methods and standards used
- Conformance of expected results
- Feasibility of system integration testing
- Feasibility of operation and maintenance

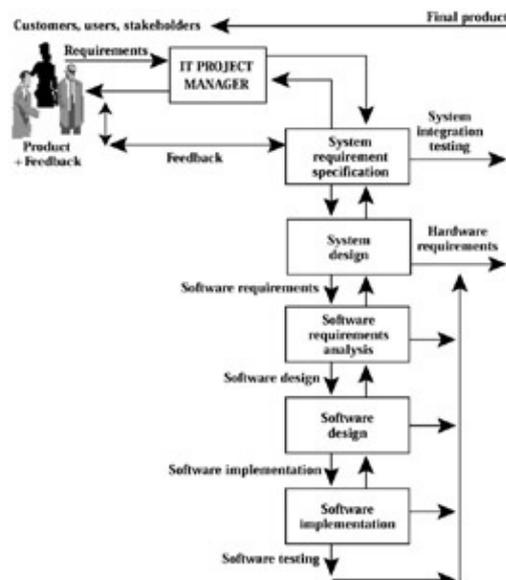


Figure 21–1: IT system integration process

Software developers install the software product in the target environment in accordance with the statement of work (SOW). The developers assist the users with the setup activities. When the installed software product is replacing an existing system, the developers support the parallel activities. The developers ensure that the software code and databases are initialized, executed, and terminated.

System Integration Testing

System developers conduct system integration testing. The IT project manager and customers witness and evaluate the results of the test. The system integration test is in accordance with the system requirements

Chapter 21: System Integration and Evaluation

specification. It ensures that the implementation of each requirement is tested for compliance and the system is ready for delivery to the customers. The developers evaluate the IT system for the following:

- Coverage of system requirements
- Conformance to expected results
- Feasibility of operation and maintenance

The following are some of the tests and analyses:

- Input–output assertions
- Structural induction
- Regression
- Transaction flow analysis
- Stress analysis
- Failure analysis
- Concurrency analysis
- Performance analysis

The input–output assertion technique uses the assertions that are associated with the entry point, exit point, and intermediate points in the source code. The composition rule of logic lets the conjunction of assertions form along particular execution paths. This confirms that if all intermediate assertions along a given execution path are true, then the truth of the input assertion implies the truth of the output assertion for the given path.

Structural induction is based on the principle of mathematic induction. This principle is illustrated as follows: Suppose there is a set S . Assume that P is true for the first element in the set S . If P is less than or equal to N for the first element in set S , then P is true for the $N + 1$ first element in set S .

Structural induction generally applies to recursive data structure. Regression is the retesting of the system to detect for errors introduced during modification. The system modification mandates that regression testing be performed. The developers must test the older capabilities of the system on which the customer is dependent rather than the new capabilities provided by the modification. The tests are supplemented with the other specified tests for the modifications; step–by–step testing of each affected component with the modification is of prime importance. This testing follows the logical flow of the system architecture. It involves retesting, starting from the lowest–level components, and combining these into the computer software configuration item (CSCI) and hardware configuration item (HWCI), which are ultimately threaded into the system.

The transition flow analysis identifies the transactions by drawing data flow after integration testing to model the logical flow of the system. The developers compare and confirm this analysis with the system architecture.

Stress analysis involves the behavior of the system when its resources are saturated to assess whether the system will continue to satisfy its specifications. The developers must identify the resources that can be stressed depending on a particular system. The following are some of the resources for stress test cases:

- File space
- Memory
- Input–output buffers
- Processing time
- Runtime
- Interrupt handler

Stress analysis forces a view of the system under unforeseen conditions that the user may not anticipate.

Case Study 21–1: Doomed Mars Spacecraft

Failure analysis is an examination of the system products reaction to various types of potential failures from hardware, software, or communication. The developers examine the products specifications to determine precisely which types of failures must be analyzed and what the products reaction must be. This process assists the developers in detecting the system products ability to recover from errors, such as lost files, lost data, and duplicate transactions.

Concurrency analysis examines the interaction of the tasks that are being simultaneously executed within the system to ensure the satisfaction of the system specifications. The developers may execute concurrent tasks in parallel or interweave their execution. The developers perform this analysis during the design phase to identify such issues as potential contention for resources, deadlock, and priorities.

A concurrency analysis for implementations takes place during system testing. The developers should design, execute, and analyze tests to exploit the parallelism in the system and ensure the satisfaction of the specifications.

Performance analysis ensures that the product satisfies its specified performance objectives. The developers apply a performance analysis during each of the IT system products V & V activities. The developers perform a performance analysis during each level of testing.

The IT project manager and customers verify the test data that are constructed to correspond to the scenarios for which the performance requirements are specified. They collect, analyze, and interpret test data for the correctness of the result against the system requirements.

Statistical methods and automated test tools are available to assist for collection and analysis of the data. Data management evaluates a system for quality assurance (QA) and effective control of data. The data management process includes the following:

- Integrated data requirement list
- Instrumentation and collection methods
- Data transmission, reduction, and storage
- Analysis and reporting
- Audit trail tracing of all data decisions

The IT project manager and customer make decisions concerning the number and duration of the tests depending on resources, budget, and schedule.

Case Study 21–1: Doomed Mars Spacecraft

Missing computer codes, inadequate software testing, and a communications systems that lacked integration helped doom the Mars Polar Lander (MPL).

After losing all four Mars-bound spacecraft in 1999, officials of the National Aeronautics and Space Administration (NASA) have had to rethink their approach to using robotic spacecraft. The reports by the Mars Program Independent Assessment Team and NASAs Jet Propulsion Laboratory (JPL) Review Board will guide future mission efforts.

The JPL study, Reports of the Loss of the Mars Polar Lander (MPL) and Deep Space 2 Missions, concluded that the MPL probably crashed because of spurious signals that the MPL generated during its descent, a problem that would have been fixed by one line of missing code. The lack of communications (telemetry) to

System Requirements Verification and Validation

provide entry, descent and landing data for [the Mars Polar Lander] was a major mistake, the report says. Absence of this information prevented an analysis of the performance of MPL and eliminated any ability to reflect knowledge gained from MPL in future missions. The reports also confirmed that the Mars Climate Orbiter failed in 1999 because navigation data were recorded in English measuring units rather than metric.

Thomas Young, a space industry executive and former NASA official, led the assessment team. The study faulted NASA for inadequate oversight, testing, and independent analysis of the orbiter project (*Government Computer News*, 2000).

System Requirements Verification and Validation

System requirements V & V is a process of ensuring that the IT system is developed correctly. Customers, users, and stakeholders independently conduct the V & V process. Verification determines whether a system development product fulfills the customers requirements. Validation evaluates and ensures that the IT system is in compliance with its requirements. The V & V process includes formal techniques to prove the correctness of the system products. The goal of the V & V process is to ensure that the system product is free of failures and meets the customers expectations. The objective of the V & V process is to determine whether the system products satisfy their requirements and specifications in areas such as the following:

- System requirements analysis
- System design
- Quality
- Safety
- Security
- Man-machine interface
- Serviceability

The V & V process confirms whether the system is correct and error free, the product is consistent with itself and other interfacing products, everything in the product is necessary, the product is complete, and it satisfies its performance specifications. The following are some of the techniques used for V & V:

- System technical reviews
- System testing
- Proof of correctness
- Simulation and prototyping
- Requirements tracing

During the requirements and specification V & V activities, developers analyze performance objectives to ensure the following:

- Completeness
- Feasibility
 - ◆ Prototyping
 - ◆ Simulation
 - ◆ Modeling
- Testability

The developers evaluate the performance requirement during the design phase to determine whether the customers requirement is satisfied. Prototyping, simulation, and modeling are applicable techniques.

Management of IT Project Documentation

The V & V process confirms the satisfaction of the customers criteria as specified in the SOW and verifies the completion of all audits, such as functional and physical configuration. The V & V process also confirms agreement on the number and severity of unresolved system documentation errors. Once the customer is satisfied, the certification of V & V is signed and the acceptance of the system engineering, source and object code, hardware, and all related documents is complete.

Management of IT Project Documentation

Management of IT project documentation depends on the agreement between the customers and the management. The guidance and formats have already been established in the SOW. The approved and released technical documentation meets the following requirements as stated in the SOW:

- Be complete
- Establish the product baseline
- Be suitable for use in the maintenance
- Be satisfactory for accepting items produced
- Be appropriate for operational needs
- Be appropriate for logistic support purposes

Customer Acceptance Testing

Customer acceptance testing involves the proper planning and execution of tests that demonstrate that the implemented system engineering for the IT system satisfies the customers requirements. The test includes functional, performance, and stress tests that were developed during the systems development testing and integration testing. For his or her own satisfaction, the customer performs the acceptance tests to ensure that the result achieved is correct. The following are some of the tools used for acceptance tests:

- Coverage analyzer
- Timing analyzer
- Standard checker
- Statistical methods

Customers, users, and stakeholders conduct independent configuration audits, which verify compliance with requirements. The audit function validates the accomplishment of development requirements and achievements of a production configuration through the commercial items (CIs) technical documentation. The following are two types of configuration audits:

- Functional configuration audit (FCA)
- Physical configuration audit (PCA)

The IT project manager conducts these audits in conjunction with other audits such as reviews, demonstration tests, inspections, and acceptance tests. Sometimes the customer, in accordance with SOW, requires these audits.

An FCA is a means of validating the development of a CI that has been completed satisfactorily. This is a correlated prerequisite to a PCA. The manager conducts an FCA on CIs to ensure that the CIs technical documentation accurately reflects the CIs functional characteristics and necessary physical characteristics, and that test and analysis data verify that the CI has achieved the performance specified in its functional or allocated configuration identification.

Management of the IT Project Audit

A PCA is a means of establishing the product baseline as reflected in the product configuration identification and is used for the production and acceptance of the units of a CI. This ensures that the as-built configuration of a unit of a CI is selected jointly by the system developer and matches that same units product configuration identification or that the differences are reconciled. It also ensures that the acceptance testing requirements prescribed by the documentation are adequate for acceptance of production units of a CI according to the QA standards. Formal approval by the IT project manager of the CI specifications and the satisfactory completion of a PCA results in the establishment of the product baseline for the configuration item.

Management of the IT Project Audit

The IT project manager starts an audit team to work immediately after the system products are validated and verified by the customers. This team is called an *external audit team*. The team generally consists of members who were not involved in the systems development. The auditor evaluates the projects success or failure and measures the organizations achievements according to the following objectives:

- Managed the system development well
- Completed the project within the schedule and budget
- Performed the project well
- Managed the project well
- Hired a competent, effective technical team
- Satisfied customers, users, and stakeholders
- Used Internet and reuse technologies effectively
- Employed modern automated tools
- Learned lessons for future IT projects

Well-Managed, Successful IT Project

Figure 21-2 illustrates the characteristics of a well-managed, successful IT project. The cost and the risk involved influence the progress of a systems development. Before the cost increases, the chance of achieving the goals of the systems development successfully decreases. The following are characteristics of a well-managed, successful IT project:

- The customers, users, and stakeholders are satisfied.
- Fewer bugs exist in the system.
- Team members are satisfied.
- The technical aspect is handled well.
- The schedule is met.
- The IT project is completed successfully within budget and time.
- The project is well documented.

Management of the IT Project Audit

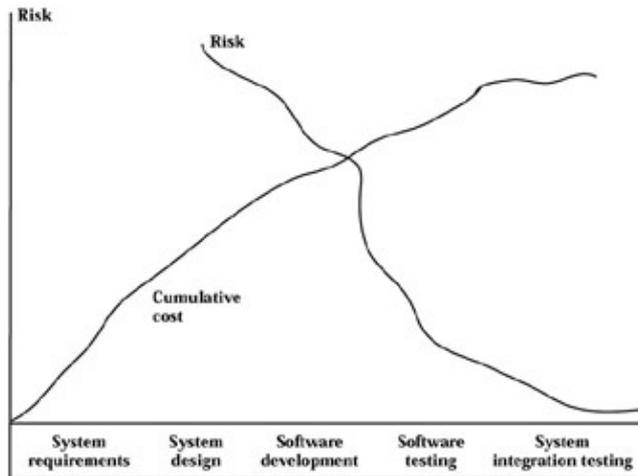


Figure 21-2: A well-managed IT project

The following is an auditors checklist for a well-managed, successful IT project:

1. Management

- a. Customers, users, and stakeholders are satisfied.
- b. The system development staff is satisfied.
- c. The system is well designed and meets the customers requirements.
- d. The system is completed on time.
- e. The system is completed within budget.
- f. The systems development is well documented.
- g. The system has a minimum level of efforts for maintenance.
- h. The IT project manager used good project management techniques.
- i. The system sets an example for future projects.

2. Technical

- a. The developers used automated computer-aided software engineering (CASE) and commercial-off-the-shelf (COTS) tools.
- b. The developers applied effective reuse.
- c. The developers used maximum Internet technology.
- d. The developers followed the team concept.
- e. The developers followed modern techniques for system development.
- f. The developers had an ongoing open communication with the project team.
- g. Supporting team assistance was readily available.

3. Operation

- a. Production execution is easy.
- b. The system has a runtime library.
- c. The system operates according to a reasonable schedule.
- d. The system has good execution documentation.
- e. The system allows for smooth conversion.
- f. The system requires a minimum man-machine interface.
- g. The system takes the users safety and security into consideration.
- h. The system is reliable and dependable.

4. Resources

Unsuccessful IT Project

- a. The system is accessible and available to the customer.
- b. The system has allocated time and budget for planning and control.
- c. The system has a realistic schedule.
- d. The system has adequate resources.
- e. The system has the customers approval.
- f. The system development team members demonstrate personal growth.
- g. The development team members have a high morale.
- h. The system has project management support.
- i. The system has management recognition.
- j. The system has management follow-up support.

Unsuccessful IT Project

Figure 21–3 illustrates characteristics of an unsuccessful IT project. The IT project failures or nonacceptance by the customer are due to many reasons. Figure 21–3 shows a poorly managed project. The determination of the unsuccessful project requires a serious audit evaluation of the following:

- Management
- System
- Technical
- Personnel
- Resources
- Operation
- Documents
- Customers, users, and stakeholders

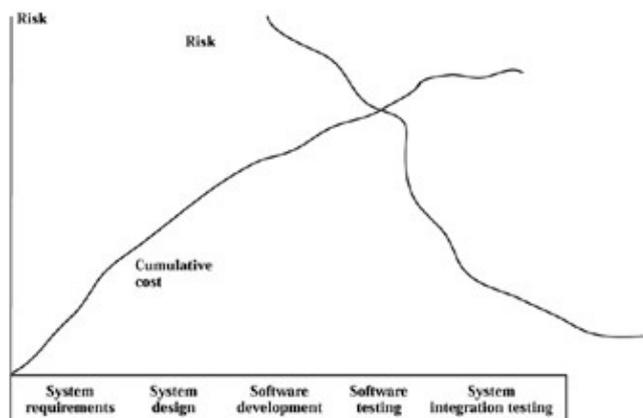


Figure 21–3: A poorly managed IT project

The IT project manager evaluates a project to determine whether lessons can be learned as a result of completing this project. The manager examines the style and techniques used during the systems development phases. The manager learns lessons by examining problems encountered in areas such as planning, manpower, budget, scheduling, estimation, and organization. He or she should apply methods that can help in avoiding the recurrence of such problems in future IT projects. The following are the main evaluation activities of the manager:

- Redetermine objective and scope.
- Review the project book.
- Review tasks and activities.
- Review assignment of responsibilities.

Unsuccessful IT Project

- Reevaluate planning, scheduling, and estimating.
- Reevaluate budgeting.
- Reevaluate risk factor management.
- Reevaluate organization.

The manager studies the system evaluation to determine whether the system was feasible, achieved its objectives, or met the customers requirements. The system evaluation also determines the following:

- Whether the system development met the design criteria with regard to solving the customers requirements
- Whether the technical aspect of the systems development met the performance objectives in terms of resources; response timing; principles; and goals of system development, safety, and security
- Whether the cost objectives were met in terms of recurring operating costs
- Whether the data flow procedural system was performed effectively for the customer
- The extent of the customers satisfaction with the systems development
- The effectiveness of the man-machine interfaces
- Whether the user guides, operator notes, and other necessary documentation are adequate to enable full and proper use of the system
- Whether the system documentation and procedures are adequate to maintain the system

The following are some of the questions that the IT project manager should ask during a technical evaluation:

- Does the technical evaluation confirm the use of proper methodologies in system and software development?
- Did the developers use modern automated tools and CI?
- Did the developers use the system modeling and simulation technique?
- Did the developers use the Internet for reuse assets?
- Did the system efficiently use the reuse technique?
- Did the developers study and analyze the customers requirements properly?
- Did the developers rush the system to meet deadlines and not debug or test it properly?

Any lesson learned can help the system developers and project manager improve future IT system development.

The manager studies the personnel evaluation to learn the following:

- Was morale high?
- Were personnel properly educated and trained to cope with the stress generated by the demands of the project?
- Were personnel properly equipped to meet the challenges?
- Were resources and support groups available?
- Did the management create tension but cooperate through open communication?

The manager uses a resource evaluation to determine the following:

- Were adequate facilities provided to complete the project?
- Did the support groups cooperate in providing the necessary functions in time?
- Were lessons learned to help in future projects?

The auditor needs to reevaluate operational manuals for comparison with other successful systems. The areas of investigation are computer operations, data entry, and interfaces with the users. During the operational

Future Trend

evaluation the auditor interviews the users and gets their feedback. The auditor investigates topics of man–machine interfaces, timing, and manpower requirements in data entry and operations. The auditor examines the hardware to find out if it has the capability to cope with the present volume of data and whether the quality of the result is satisfactory. The auditor reviews the systems performance, controls, equipment use, and reliability. The operation logs and the trouble reports show problems such as restarts, restoring databases, reruns due to errors, and error and switch–handling capabilities.

A document evaluation is an audit review of all of the available documentation. This includes documentation of the systems development, reviews, manuals, operational information, and correspondence between the customer and developers. This familiarizes the auditors with the project and enables them to pinpoint specific areas for investigation.

The customers evaluation is of prime importance. The auditor interviews the customer to determine the following:

- Did the customer perceive the system as successful?
- Did the customer alert the developers at any time that the requirements were not understood?
- Was the customer involved in all phases of the systems development?
- Did the customer participate in all reviews?
- Did the customer show dissatisfaction?
- Did the developers take necessary actions and efforts to satisfy the customer?
- Did the customer approve the action taken by the developer?
- Did the customers and developers have effective communications?
- What lessons can be learned to avoid problems that may occur in the future?

The auditors write reports after completion of the findings and submit them to the project manager with their recommendations. The report should be precise and to the point with supporting facts and figures. The following is an outline of the evaluation report:

1. Introduction
 - a. Purpose and objectives
 - b. Scope
2. Management summary
 - a. Major findings
 - b. Recommendations
3. Measurement of project management performance
 - a. Budget (planned versus actual cost)
 - b. Schedule (planned versus actual completion dates)
4. Evaluation section
5. Recommendations
6. Conclusion
7. Appendixes and exhibits

Future Trend

The future trend in the computer industry is to follow the successful footsteps of other engineering faculties. The phrases Reuse, reuse, reuse and Model, simulate, fix, test, iterate are followed to save time and cost as has

Checklist: Suggestions for a Successful IT System Project

been achieved in other branches of engineering. Understanding the use of modeling and simulation across the total life cycle of the IT system with the process of requirements generation, project management, design and system development, testing and evaluation, and training is key to reduction of schedule and cost. It also enables testing of the performance of the system in numerous scenarios to provide confidence to the management. The project manager learns lessons from the computer hardware industry that through practice of reuse and modeling he or she can reduce pricing by reusing architecture and components. For example, when building a house or factory, the manager does not invent new products but reuses what is available in the market to save time and cost.

A lot of information is available on the Internet. The manager should try to extract as much knowledge as possible via the Internet and reuse it. More research is being conducted to make the Internet efficient in handling visual information.

A need also exists to capture a persons experience when he or she retires. This valuable knowledge is wasted when a person retires after gaining valuable years of experience, taking the knowledgeable data with him or her. Electronic chips, knowledge-based system software, and continuous feedback for improvement do not need to be reinvented. A software mechanism is necessary to capture that valuable knowledge to be reused by others who will perform the same types of duties in the future. Instead of starting from the beginning, a new person can gain the experienced knowledge and avoid going through the same training. This process can save cost and time, make a new person on the job more experienced, and share valuable contributions in a short time. This knowledgeable data can be populated on the personal domain specific repository to be reused in the future. This system can be reused at various levels and among various different fields of life and industry.

A common saying in the industry is that a computer is as good as its design by reuse of well-tested hardware components. Computer software is also as good as its design by reuse of well-tested software components as provided by the domain engineer. They all work to save cost and time and generate a quality product.

On the Internet, wireless technology is everywhere and everything. The World Wide Web is definitely going wireless. Many companies are showing off technologies that allow the Internet to be brought anywhere and accessed through pagers, handheld devices, and cellular phones. People will be trading over the Internet, getting information, and managing their bills. New services allow consumers to receive and pay bills using a phone enabled by wireless application protocol (WAP) or a Palm VII handheld device. Desktop services also allow consumers to view summary reports and update data to money management programs such as Quicken. The system works with electronic bills and traditional paper bills, which the company scans into a computer. One reason that wireless is so exciting is that it increases the ability to be in touch with customers.

Checklist: Suggestions for a Successful IT System Project

The checklist describes why the IT system project is a victim and how to turn around a failing project. Experience shows that most of the system projects become victims of failure for the following reasons:

- The system development team did not understand the requirements of the customers, stakeholders, and users.
- The scope of the project is not well defined.
- The chosen software and hardware technology changes.
- The project lacks senior management support and commitment.
- The return on investment (ROI) is not realistic.
- The schedule for completing the project is not real.
- The users are resistant because of proprietorship of assets.

Checklist: Suggestions for a Successful IT System Project

- The project lacks professionals with appropriate skills.
- The project lacks cooperation and enthusiasm among the practitioners.
- Fear exists that the success of the project is going to hurt the future business.
- Industry best practices and lessons learned are ignored.
- The project has frequent changes in manpower and management.
- Self-egotism and self-pride play a major part in hurting the project.
- The project was incorrectly presented to senior management.
- Management has a prejudice to dogmatic views.
- Customers, stakeholders, and users are not involved in the progress of the project, creating a vacuum and resistance.
- Professionals are not paid for completing the project; rather, they are compensated for the number of hours billed.
- A nonseasonal manager is assigned to manage a highly visible project.
- The project team has communication problems.
- Miscommunication exists among the key players, management, customers, stakeholders, and users.
- Reuse benefits are not properly communicated among the practitioners.
- The budget dries out.
- Milestones are not achieved on schedule.
- The project is not advertised properly.
- The testers are not involved in the beginning of the project.
- The systems QA and CM sections are not established early enough and involved in the beginning of the project.
- The IT system is not tested well.

Chapter 22: Practical Case Study

This case study is based on an actual IT program. The consultant was contracted to develop the management information system (MIS) program methodology and reporting system. Despite having established new IT program management guidelines, this program followed the old methodologies and employed the traditional mindset encountered in typical IT programs.

Company Background

This national corporation had been purchased twice within a 3-year period, with all employees and assets included in the transaction. During the second takeover, many of the IT staff members took positions with other firms in the area. The parent corporate MIS department hired a new Chief Information Officer (CIO) to head the MIS organization. He in turn brought in a new senior manager to assist in overseeing the MIS staff. Simultaneously, the corporate MIS department hired someone from outside to be the CIO, but he was relegated to the position of vice president (VP) of MIS at the division level.

Corporate MIS Structure

The corporate structure consisted of the parent corporation divided into several divisions. A chief executive officer (CEO) and his staff headed each division. On each staff was a CIO who ran the MIS department. At the parent level was an MIS department that set the overall strategy for the divi-

sions. The parent MIS department provided service professionals who were responsible for help desk support, equipment configurations and installation, maintenance, systems support, network support, creation of new user accounts, monitoring of software usage, and maintenance of the MIS inventory. These professionals were assigned to a division full time but were paid and evaluated by the parent MIS department.

At the division level, the CIO reported to the CEO and separated his organization into two major departments:

1. The Software Development and Maintenance department was responsible for new application software development and the maintenance and support of existing software applications, training, and customer services (mini help desk). The new senior manager of MIS, brought in by the CIO, headed this organization.
2. The Business Development and Technical Services department consisted of a consulting organization, technical support and development, and business analysis. The VP of MIS was brought in by the corporate managers who headed this department.

Conflict

The CIO and his manager had no experience in this industry, whereas the VP of MIS came from a similar company and had a vast amount of experience. Soon two distinct factions evolved within the MIS staff, and the board of directors (BOD) became heavily entrenched in trying to sort out which faction was pointing them in the right direction (Figure 22-1). The CIO did not want his authority compromised and continued to ensure the board that all was well and on schedule.

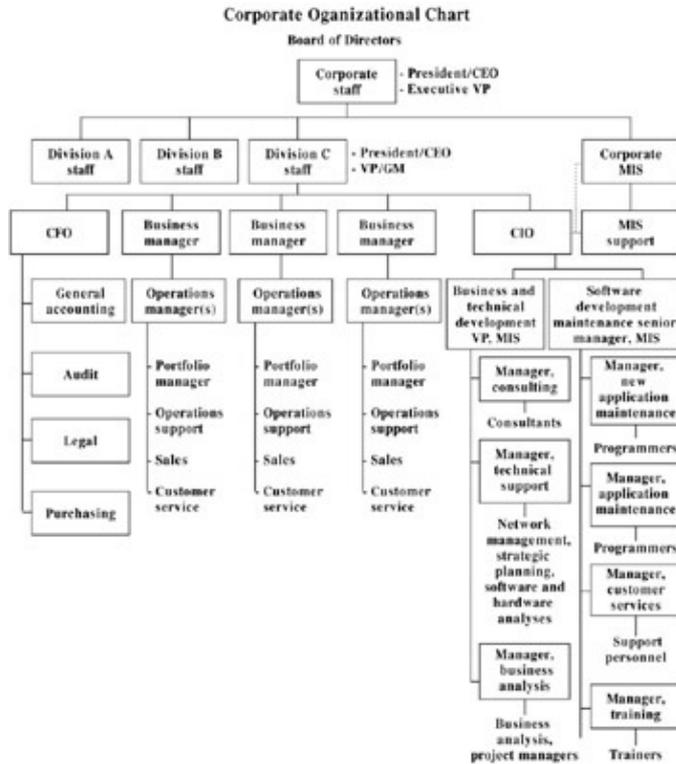


Figure 22–1: Organizational chart

Hardware and Network Configuration

The user hardware consisted of personal computers (PCs) running on Novell local area networks (LANs). PCs acting as router hubs connected each LAN to the central database. The corporate service organization mandated that all users would use a Compaq 486 PC with Intel 33 MHz chips. The router hubs were Compaq 386 PCs with Intel 33 MHz chips. The database consisted of approximately 30 gigabytes of storage on a disk farm.

Software Environment

Most of the existing applications were developed in Clipper, an outdated suite of development software. New applications were developed in

Sybase and Powerbuilder. No documentation of the Clipper environment existed, and seasoned professionals, mostly contractors, were assigned to support that application set. Tenured staff members were used to supervise the remaining contractors doing development in Sybase and Powerbuilder.

Corporate Service Professionals

Corporate service professionals provided a variety of services to the division. During one period of growth, while hiring some new contractors, a need arose for additional PCs. The division argued that the services group should order new 66 MHz models to improve productivity in the MIS area. When the new PCs arrived, the application development area improved, but the greatest influence was how fast the user applications ran on the new hardware. The users were brought in to see the improvements in response time and readily agreed to purchase the newer models.

Role of Business Services and Technical Support

However, when MIS went to the service group to place the order for newer models, they were told that inventory control practices for maintenance and spare parts prohibited purchase of newer models until the corporate approval was given.

As new contractors and staff members came on board, they would set up their new accounts by copying applications from the application database server. The support group would monitor the number of licenses being used and, if the quota were exceeded, would deny access to the software. The contractors were expensive to employ and were kept waiting until budgets, approvals, and purchasing and license agreements were settled. This often took several days to resolve.

Role of Business Services and Technical Support

The technical support organization was evaluating operating systems for future MIS applications in the division. Windows NT and Windows 95 were among the candidates. However, they were primarily interested in the technical aspects and seemed unconcerned with the ramifications of software availability for these new operating systems. No transition plan existed to migrate existing applications, nor was there availability of development software on the new platform.

In the meantime, the consultants and business analysts were interfacing with the user community, defining new applications to be built by the software development group. They busily established project schedules and prioritized the development efforts. When the software development group hesitated and pushed back the schedule because of personnel shortages, chaos erupted in the user community.

Role of Software Development and Maintenance

This organization was under heavy user pressure. Peak periods were during the day when the users were constrained to a specific amount of time to process their work. Failure to do so resulted in extreme customer pressure and missed deadlines. The equipment in the Novell network was so outdated that the users began experiencing severe gridlock in the network. New applications that were recently installed had to be halted, and the users had to go back to manual or old methods of performing their work.

More chaos erupted, and everyone was fighting fires. Experiments were made with the router hubs; the changes decreased performance and made the MIS department look as if they did not know what they were doing. They reverted back to the old setup and promised to correct the situation. To complicate matters, members of the MIS staff continued to break ranks, and soon the contractors outnumbered the permanent staff. The lack of tenured staff members had everyone fighting fires, rescheduling program and project implementations, and working overtime. During this period of conflict, the CIO committed to the BOD that a new program, urgently required by one of the user groups, would be implemented on schedule and within budget.

Program

This program was initially set up using the new program management guidelines and predicted an implementation schedule that would last 6 months. The integrated program schedule was documented using Microsoft Project, in which fully loaded costs were used to determine program costs. Figure 22–2 shows the program schedule, and Figure 22–3 outlines the program costs.

Program Execution

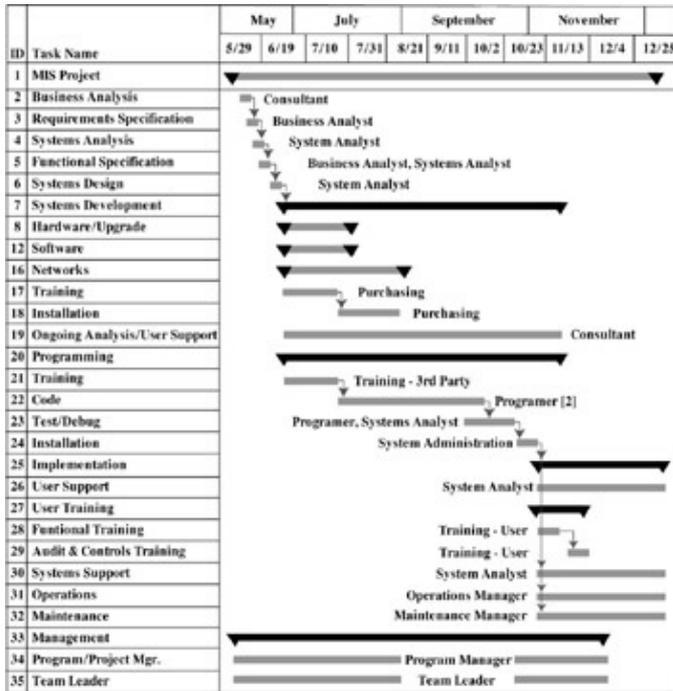


Figure 22–2: IT project schedule

ID	Task Name	Fixed Cost	Total Cost	Baseline	Variance	Actual	Remaining
34	Program/Project Leader	50.00	\$ 62,000	\$ 62,000	\$0.00	\$0.00	\$ 62,000
35	Team Leader	50.00	\$ 62,000	\$ 62,000	\$0.00	\$0.00	\$ 62,000
22	Code	50.00	\$ 48,000	\$ 48,000	\$0.00	\$0.00	\$ 48,000
19	Ongoing Analysis/ User Support	50.00	\$ 36,000	\$ 36,000	\$0.00	\$0.00	\$ 36,000
23	Test/Debug	50.00	\$ 12,000	\$ 12,000	\$0.00	\$0.00	\$ 12,000
26	User Support	50.00	\$ 12,000	\$ 12,000	\$0.00	\$0.00	\$ 12,000
30	Systems Support	50.00	\$ 12,000	\$ 12,000	\$0.00	\$0.00	\$ 12,000
31	Operations	50.00	\$ 12,000	\$ 12,000	\$0.00	\$0.00	\$ 12,000
32	Maintenance	50.00	\$ 12,000	\$ 12,000	\$0.00	\$0.00	\$ 12,000
21	Training	50.00	\$ 4,000	\$ 4,000	\$0.00	\$0.00	\$ 4,000
5	Functional Specification	50.00	\$ 4,000	\$ 4,000	\$0.00	\$0.00	\$ 4,000
2	Business Analysis	50.00	\$ 3,200	\$ 3,200	\$0.00	\$0.00	\$ 3,200
6	System Design	50.00	\$ 2,000	\$ 2,000	\$0.00	\$0.00	\$ 2,000
4	Installation	50.00	\$ 2,000	\$ 2,000	\$0.00	\$0.00	\$ 2,000
28	Functional Training	50.00	\$ 2,000	\$ 2,000	\$0.00	\$0.00	\$ 2,000
3	Requirements Specification	50.00	\$ 1,600	\$ 1,600	\$0.00	\$0.00	\$ 1,600
4	System Analysis	50.00	\$ 1,600	\$ 1,600	\$0.00	\$0.00	\$ 1,600
18	Installation	50.00	\$ 1,600	\$ 1,600	\$0.00	\$0.00	\$ 1,600
17	Training	50.00	\$ 1,600	\$ 1,600	\$0.00	\$0.00	\$ 1,600
1	MIS Project	50.00	\$ 1,400	\$ 1,400	\$0.00	\$0.00	\$ 1,400
		50.00	\$275,000	\$275,000	\$0.00	\$0.00	\$275,000

Figure 22–3: IT project costs

The program schedule was reviewed with the CIO and his staff before presentation to the BOD. Cost and schedule were reviewed, and it was determined that the cost savings would net a return on investment (ROI) of approximately 35%. The program would pay for itself in less than 3 years. The total cost estimate was approximately \$275,000. The program team consisted of a program manager, team leader, consultant, business analyst, systems analyst, contract programmers, MIS operations, and trainers. Not all members of the team were assigned full-time responsibilities.

Program Execution

The program started off well. Gathering data for the functional and technical requirements went smoothly, and the user group was satisfied. The program team went right to work on the screen design for the user interface and proceeded to change record structures to accommodate the new data fields. The analysts and programmers were delighted with the new functionality that they could provide to the users and eagerly showed everyone in

Results

the MIS department their progress.

The user group would have a vast array of options from which to choose, and this would enable them to provide better service to the customer base. Trainers joined the team and proceeded with the same vigor as the analysts and programmers. They developed an impressive training package that showed that they were proud of their accomplishments. During this transition, the CIO was reporting to the BOD that all was well and on schedule. The BOD members representing the user groups took it for granted that the MIS was keeping their management abreast of the program content as it progressed. However, they did not realize that the user group had little free time to spend with their MIS counterparts. The program team thought that this was business as usual and usually did not consult with the user group on screen layouts or option functionality.

Training was scheduled and started without any member of the user group having seen the new screen designs or the supposed improvements in functionality. When the actual training sessions began, the users were disappointed with the software. This vast array of options required them to plod through several screens that were too time consuming. The user group wanted to speed up the process so that more work could be done in shorter period of time and consequently gain improvements in customer satisfaction.

Results

Many problems faced the MIS organization. Some had to do with hardware and personnel resources. Other problems arose out of politics and job security. The program was never completed, and the software was not installed. The users flatly refused to accept it. The functionality was not what the users wanted, and a software rewrite would take too long. More important things were now the focus. The capital influence on ROI was about \$375,000. More importantly, the user community experienced staff increases and constant pressure as the customer base increased along with the proportional workload.

User confidence in the MIS department had eroded, but a significant backlog of work still had to be accomplished. Somehow, the MIS management structure remained in place, and they continued to make promises they could not keep. The management and their staff continued to spend significant amounts of money on contractors, consultants, and enabling technology that did not blend into a coherent solution for the end-users.

Lessons Learned

Considering the documented history of this program, it was doomed from its inception. Too much time was spent on politics and keeping management in the dark concerning the true nature of the problems facing the MIS department. The company became caught up in the everyday process of putting out the next fire. Someone, somehow, should have evaluated the risks associated with this program and other programs that were under development at the time. Assessment of the companys position relative to the coordination of resources, technical direction, staffing shortfalls, and lack of user involvement would have alerted someone that the project was not going well.

Risk management is a difficult process. It is not a process of excuses but of identifying the obstacles to successful completion of an assigned task. The difficulty is in the assessment of its influence in terms of cost, schedule, and probability. The more difficult task is the realization that a good program manager should have alternate solutions to abate the risk and identify how the cost, schedule, and probable influence on a program schedule could be reduced.

Risk Analysis

Risk can be associated with a root cause. A risk abatement strategy can be formulated by bypassing the symptoms and focusing on the root cause. Associated risks can be minimized through identification of several risks that have the same root cause and elimination of the root cause.

In the program discussed in this case study, the same root cause (changing requirements) was documented for seven of the risks identified. The potential cost damage of these risks was in excess of \$100,000. Elimination of the root cause by establishment of the baseline configuration would diminish the probability of such an extravagant cost overrun. Several instances existed in this program of multiple risks associated with one root cause. A few fixes could have solved many problems and may have led to successful completion of a difficult program in adverse circumstances. For example, the lack of human resources was cited three times and carried a potential cost of \$180,000. Mitigation of this risk by the proper allocation of personnel could have saved money and increased the projects chance for success.

The program should have been on hold until these problems were resolved. The MIS department could have saved its reputation, the user community would have been alerted to its own internal requirements, and the corporation would not have made a bad investment of time, resources, and capital.

Using RiskTrak, the consultant spent a couple of hours identifying the risks associated with this program and honestly evaluating the influence in terms of cost and schedule. The surprise came when the reports were generated and the indication of potential cost and schedule influence was realized. After careful consideration, the consultant realized that using a risk management tool could have saved the corporation several hundred thousand dollars during his 6-month tenure across a variety of programs. The cost and schedule reports indicated that not all of the potential damages to cost and schedule would have occurred. The risk management software factors in the percentage of probability and damage to forecast predictable results if the risks identified are not managed. The risk-identification process triggered and freed the program managers mind from only considering the risks associated with the most recent program.

If this large corporation had used the RiskTrak preproject analysis before making a decision to proceed with this project, they would not have launched the program. For a potential ROI of less than \$100,000 per year, the risk in terms of cost, time, and schedule clearly outweighed the gain, especially when considering a possible schedule slip of approximately 18 months and a cost overrun in excess of \$375,000 on a 6-month, \$275,000 project. It would have taken over 2 years to develop this project and over 6 years to retire the investment, which would not make it a cost-effective project. If the RiskTrak risk engineering had been applied and only the top two risks (e.g., changing requirements and lack of human resources) were identified and effectively managed, a potential slip of 260 days could have been avoided and over \$180,000 could have been saved. Although this would still leave a substantial cost and schedule overrun, the potential for additional savings existed in which further risk management could have been applied to mitigate the potential losses in time and money. RiskTrak postmortem analysis shows that poor planning and inconsistent data were key to the failure of this project (from Services and Technology Group, www.stgrp.com).

Acronyms

A-C

ACAP

Analyst Capability

AEXP

Risk Analysis

	Application Experience
<i>AI</i>	Artificial Intelligence
<i>ANSI</i>	American National Standards Institute
<i>API</i>	Application Program Interface
<i>ATD</i>	Actual to Date
<i>CASE</i>	Computer–Aided Software Engineering
<i>CCB</i>	Configuration Control Board
<i>CCSOM</i>	Computer Center Software Operational Manual
<i>CDR</i>	Critical Design Review
<i>CDRL</i>	Contract Data Requirements List
<i>CEO</i>	Chief Executive Officer
<i>CFD</i>	Control Flow Diagram
<i>CFO</i>	Chief Financial Officer
<i>CI</i>	Commercial Item
<i>CIO</i>	Chief Information Officer
<i>CLIN</i>	Contract Line Item Number
<i>CM</i>	Configuration Management
<i>CMM</i>	Capability Maturity Model
<i>COCOMO</i>	Constructive Cost Model
<i>COM</i>	Common Object Model
<i>CORBA</i>	Common Object Request Broker Architecture
<i>COTS</i>	Commercial–Off–the–Shelf
<i>CPM</i>	Critical Path Method
<i>CPU</i>	Central Processing Unit
<i>CSC</i>	Computer Software Component
<i>CSCI</i>	Computer Software Configuration Item

<i>CSDM</i>	Computer Software Development Methodology
<i>CSOM</i>	Computer System Operator Manual
<i>CSU</i>	Computer Software Unit

D

<i>DADP</i>	Domain Analysis and Design Process
<i>DBDD</i>	Database Design Document
<i>DBMS</i>	Database Management System
<i>DCE</i>	Distributed Computing Environment
<i>DCI</i>	Distributed Computing Infrastructure
<i>DCOM</i>	Distributed Common Object Model
<i>DD</i>	Data Dictionary
<i>DDD</i>	Domain Data Dictionary
<i>DDL</i>	Data Definition Language
<i>DDM</i>	Domain Dynamic Model
<i>DE</i>	Domain Engineering
<i>DFD</i>	Data Flow Diagram
<i>DFM</i>	Domain Functional Model
<i>DID</i>	Data Item Description
<i>DII</i>	Dynamic Invocation Interface
<i>DIM</i>	Domain Information Model
<i>DOD</i>	Department of Defense
<i>DOE</i>	Distributed Object Environment
<i>DOI</i>	Distributed Object Infrastructure
<i>DOM</i>	Distributed Object Model

<i>DOT</i>	Distributed Object Technology
<i>DPM</i>	Domain Prototype Model
<i>DSI</i>	Delivered Source Instructions
<i>DSOM</i>	Distributed System Object Model
<i>DSSA</i>	Domain-Specific Software Architecture

E-I

<i>EAC</i>	Estimate At Completion
<i>EC</i>	Estimated Cost
<i>ECP</i>	Engineering Change Proposal
<i>EIA</i>	Electronic Industries Association
<i>ELOC</i>	Estimated Line of Code
<i>ER</i>	Entity Relationship
<i>EV</i>	Earned Value
<i>FCA</i>	Functional Configuration Audit
<i>FP</i>	Function Points
<i>FQT</i>	Formal Qualification Testing
<i>GUI</i>	Graphic User Interface
<i>HTML</i>	Hypertext Markup Language
<i>HW</i>	Hardware
<i>HWCI</i>	Hardware Configuration Item
<i>ICAM</i>	Integrated Computer-Aided Manufacturing
<i>ICT</i>	Intelligent CASE Tools
<i>IDD</i>	Interface Design Document
<i>IDEF</i>	Cam DEFinition

<i>IDL</i>	Interface Definition Language
<i>IEC</i>	International Electro–technical Commission
<i>IEEE</i>	Institute of Electrical and Electronics Engineers
<i>IIOF</i>	Internet Inter–ORB Protocol
<i>I/O</i>	Input/Output
<i>IORL</i>	Input/Output Requirements Language
<i>IPT</i>	Integrated Product Team
<i>IRS</i>	Interface Requirements Specification
<i>ISO</i>	International Standards Organization
<i>IT</i>	Information Technology
<i>IV&V</i>	Independent Verification and Validation

L–O

<i>LOC</i>	Lines of Code
<i>LOE</i>	Level of Efforts
<i>MANPRINT</i>	Manpower and Personnel Integration
<i>MIS</i>	Management Information System
<i>MM</i>	Man Month
<i>MMI</i>	Man–Machine Interfaces
<i>NATO</i>	North Atlantic Treaty Organization
<i>NDI</i>	Nondevelopment Items
<i>NDS</i>	Nondevelopment Software
<i>OAM</i>	Object Analysis Model
<i>OBM</i>	Object Behavior Model
<i>OCD</i>	Operational Concept Document

<i>ODF</i>	Object Definition Language
<i>OIM</i>	Object Information Model
<i>OLE</i>	Object Linking and Embedding
<i>OMA</i>	Object Management Architecture
<i>OMG</i>	Object Management Group
<i>OML</i>	Object Management Library
<i>OO</i>	Object Oriented
<i>OOD</i>	Object-Oriented Design
<i>OODB</i>	Object-Oriented Database
<i>OODBMS</i>	Object-Oriented Database Management System
<i>OODM</i>	Object-Oriented Design Method
<i>OOM</i>	Object-Oriented Methodology
<i>OOP</i>	Object-Oriented Programming
<i>OOSD</i>	Object-Oriented Structured Design
<i>OPM</i>	Object-Process Model
<i>OSF</i>	Open Software Foundation
<i>OSI</i>	Open System Interconnection
<i>OTS</i>	Off-the-Shelf

P-S

<i>PC</i>	Personal Computer
<i>PCA</i>	Physical Configuration Audit
<i>PERT</i>	Program Evaluation and Review Technique
<i>PIN</i>	Personal Identification Number
<i>PSM</i>	Practical Software Measurement

<i>P-Spec</i>	Process Specification
<i>QA</i>	Quality Assurance
<i>QAS</i>	Quality Assurance Section
<i>RDM</i>	Requirements Definition Model
<i>RE</i>	Re-software Engineering
<i>RELY</i>	Required Software Reliability
<i>RFP</i>	Request for Proposal
<i>ROM</i>	Read-Only Memory
<i>RSE</i>	Reverse Software Engineering
<i>RSO</i>	Reusable Software Objects
<i>RT</i>	Requirements Tracer
<i>RTE</i>	Run-Time Environment
<i>RTL</i>	Run-Time Library
<i>SA</i>	Structured Analysis
<i>SCR</i>	Software Cost Reduction
<i>SD</i>	Structured Design
<i>SDD</i>	Software Design Document
<i>SDF</i>	Software Development File
<i>SDL</i>	Software Development Library
<i>SDP</i>	Software Development Plan
<i>SDR</i>	System Design Review
<i>SECP</i>	Software Engineering Conversion Plan
<i>SED</i>	Software Engineering Design
<i>SEDD</i>	System Engineering Design Document
<i>SEDP</i>	Software Engineering Development Plan
<i>SEI</i>	

	Software Engineering Institute
<i>SEMP</i>	Software Engineering Maintenance Plan
<i>SERA</i>	Software Engineering Requirements Analysis
<i>SII</i>	Static Invocation Interface
<i>SIOM</i>	Software Input/Output Manual
<i>SIP</i>	Software Installation Plan
<i>SLOC</i>	Source Lines of Code
<i>SOM</i>	System Object Model
<i>SOW</i>	Statement of Work
<i>Spec</i>	Specification
<i>SPM</i>	Software Programmers Manual
<i>SPS</i>	Software Product Specification
<i>SRP</i>	Software Reuse Plan
<i>SRR</i>	System Requirements Review
<i>SRS</i>	Software Requirements Specification
<i>SSA</i>	Structured System Analysis
<i>SSD</i>	Strategies for System Development
<i>SSDD</i>	System/Segment Design Document
<i>SSP</i>	Software Support Plan
<i>SSR</i>	Software Specification Review
<i>SSS</i>	System/Segment Specification
<i>STD</i>	State Transition Diagram
<i>STR</i>	Software Test Report
<i>STT</i>	State Transition Table
<i>SUM</i>	Software Users Manual
<i>SW</i>	Software

SysDD System Design Document
SysRS System Requirement Specification

T-X

TRR Test Readiness Review
UC User Cases
UI User Interface
VDD Version Description Document
V & V Verification and Validation
WBS Work Breakdown Structure
WWW World Wide Web
XML eXtensible Markup Language

References

- Barr Z: Earned value analysis: a case study, *PM Network* 10:33, 1996.
- Baumert JH et al: *Software measures and the capability maturity model*, CMU/SEI 92-TR-25, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1992.
- Boehm B: A spiral model of software development and enhancement, *IEEE Computer* 21(5), 1988.
- Brooks FP, Jr: No silver bullet: essence and accidents of software engineering, *IEEE Computer* 15(1):10, 1987.
- Carleton AD et al: *Software measurement for DoD systems: recommendations for initial core measures*, CMU/SEI-92-TR-19, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1992.
- Chen M, Han J, Yu P: Data mining: an overview from a database perspective, *IEEE Transactions on Knowledge and Data Engineering* 8(6), 1996.
- Daich, GT et al: Metrics tools: Size, *CrossTalk* April 1995.
- DBMS client/server computing*, An M & T Publication, January 1993.
- Druyun DA: Acquisition policy 93M-017, Software Metrics Policy, Office of the Deputy Assistant Secretary of the Air Force (acquisition), February 16, 1994, *CrossTalk* April 1994.
- Easterbrook et al: *Experiences using formal methods for requirements modeling*.
- Erickson DR et al: Metrics tools: effort and schedule, *CrossTalk* March 1995.
- Fellenstein SJ: Year 2000 predictions for ISO 9000, *QM*, July/August 1999, p. 12.
- Firth R et al: *A guide to the classification and assessment of software engineering tools*, CMU/SEI-87-TR-10.
- Fleming QW: *Cost/schedule control systems criteria: the management guide to C/CSCSC*, Chicago, 1992, Probus Publishing.
- Fleming QW, Koppelman JM: The earned value body of knowledge, *PM Network* 10:11, 1996b.
- Florac WA et al: *Software quality measurement: a framework for counting problems and defects*, CMU/SEI-92-TR-22, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, September 1992.
- Ganska R, Grotzky J, Rubinstein J, Van Buren J: *Requirements engineering and design technology report*, software technology support center, Hill Air Force Base, October 1995.
- Gartner Group, *ADA Research Note*, November 1998.
- Gause DC, Weinberg GM: *Exploring requirements quality before design*, New York, 1989, Dorset House Publishing.

References

Giles AE et al: Metrics tools, *CrossTalk* February 1995.

Government Computer News, April 3, 2000, p. 6.

Hammer T, Huffman L, Rosenberg L, Wilson W, Hyatt L: *Requirement metrics for risk identification*, Software Engineering Laboratory Workshop, GSFC, 1996.

Hammer T: *Measuring requirement testing*, 18th International Conference on Software Engineering, 1997.

Hammer T: *Automated requirements management beware how you use tools*, 19th International Conference on Software Engineering, 1998.

Hammond E: Making order out of chaos, *Federal Computer Week*, November 15, 1999, p. 26.

Hansen GW, Hansen JV: *Database management and design*, New York, 1992, Prentice Hall.

Hatfield MA: The case for earned value, *PM Network* 10:25, 1996.

Hayden D: Picking a wireless data provider, *Field Force Automation*, April 2000, p. 32.

IEEE Std 729–1983: *Standard glossary of software engineering terminology*, February 1983.

IEEE Std 830–1993: *Recommended practice for software requirements specifications*, December 1993.

Kerzner H: *Project management: a systems approach to planning, scheduling, and controlling*, New York, 1995, Van Nostrand Reinhold.

Kitchenham B, Pfleeger SL: Software quality: the elusive target, *IEEE Software* 13(1):12, 1996.

Krasner N: *Homing in on wireless location*, *Communication*, June 1999, p. 19.

Maintainability and Quality Assurance, 1989.

Marsan CD: SuperNet set to launch, *Network World*, August 16, 1999.

Martin J: *Information Manifesto*.

NASA–STD–2100–91: *NASA software documentation standard*, NASA Headquarters Software Engineering Program, July 29, 1991.

NASA: Software assurance guidebook, NASA Goddard Space Flight Center Office of Safety, Reliability. In Ewald A, Roy M: Distributed objects, *Object Magazine* Jan 1997.

North K: Database developer, *WEB–Techniques*, June 1999, p. 18.

Object Management Group, *What is CORBA?* August 1997.

Palmer JF: *Integrating the structured techniques with JAD: leveled systems development*. A working paper presented at 12th Structured Methods Conference, August 1987.

References

- Parnas DL: On the criteria to be used in decomposing systems into modules, *Communications of the ACM* 15, December 1972.
- Parnas DL, Clements PC, Weiss DM: The modular structure of complex systems, *IEEE Transactions on Software Engineering* SE-11(3), March 1985.
- Peterson JA: *Design for Windows managers*, Technical Report, MCT, 1986.
- Peterson S: Coming to terms with software reuse terminology: a model-based approach, SEI, *ACM Software Engineering Notes*, April 1991.
- Porter AA, Votta LG Jr, Basili VR: Comparing detection methods for software requirements Inspections: a replicated experiment, *IEEE Transactions on Software Engineering* 21(6):563, 1995.
- Radosevich L: Measuring up, *CIO Magazine*, September 15, 1999a, p. 52.
- Radosevich L: Tale of a turnaround, *CIO Magazine*, September 15, 1999b, p. 56.
- Roedler G: Applying PSM to Systems Engineering, *Insight: The Armys Software Metrics Newsletter*, Winter 1999.
- Ross DT: Douglas Ross talks about structured analysis, *IEEE Computer* 18(7):80, 1985.
- Royce WW: Managing the development of large software systems, Proceedings of the 9th International Conference, *Software Engineering, IEEE Computer Society* 328, 1987.
- Sathi A, Morton T, Roth S: Callisto: an intelligent project management system, *AI Magazine* 7(5):34, 1986.
- Scacchi W: Models of software evolution, life cycle and process, *SEI Curriculum Module SEI-CM-10-1.0*, October 1987.
- Shlaer S, Mellor SJ: *Object-oriented systems analysis, modeling the world in data*, Englewood Cliffs, NJ, 1988, Yourdon Press.
- Shlaer S, Mellor S: *Object-lifecycles: modeling the world in states*, New York, 1992, Prentice-Hall.
- Siegel J: *CORBA fundamental and programming*, Philadelphia, 1996, John Wiley & Sons.
- Singletary N: Whats the value of earned value? *PM Network* 10(28):30, 1996.
- Sodhi J, Sodhi P: *Software reuse*, New York, 1999, McGraw-Hill.
- Sodhi J, Sodhi P: *Object-oriented methods for software development*, New York, 1996, McGraw-Hill.
- Sodhi J: *Software requirements, analysis, and specification*, New York, 1992, McGraw-Hill.
- Sodhi J: *Software engineering methods, management, and CASE tools*, New York, 1991, McGraw-Hill.
- Sodhi J: *Computer systems techniques: development, implementation, and software maintenance*, PA, 1990, TAB Books.

References

- Sodhi J: *Managing ADA projects using software engineering*, PA, 1990, TAB Books.
- Sodhi J: *Evaluation of teaching SERA*, Seventh Annual National Conference for ADA Technology, 1989.
- Sodhi J, George KM: *Objects with multiple representations in ADA*, Seventh Annual National Conference for ADA Technology, 1989.
- Sodhi J: Overview of ADA features for real-time systems, *Defense Science*, November 1988.
- Software*, A Sentry Publication, January 1992.
- Software*, A Sentry Publication, May 1992.
- Software*, A Sentry Publication, September 1992.
- Software Development*, A Miller Publication, 1997
- Software Development*, A Miller Publication, January 1998
- Software Engineer's Reference Book 16*, 1991.
- Software Magazine*, Entering ORB-it, January 1997.
- Sommerville I: *Software engineering*, ed 4, Wokingham, England, 1992, Addison-Wesley.
- Sommerville I, Sawyer P: *Requirements engineering a good practice guide*, Philadelphia, 1997, John Wiley & Sons.
- Spurr K et al: *Software assistance for business re-engineering*, Philadelphia, 1994, John Wiley & Sons.
- Stikeleather J, Clarke J, Fingar P: *Distributed computing for business*, Technical Resource Connection.
- Stokes DA: Requirements analysis, *Computer Weekly Software Engineers Reference Book 16*, 1991.
- Stokes DA: Requirements analysis, *Computer Weekly*.
- The Sunday Oklahoman*, April 23, 2000.
- USA Today*, January 2000.
- Van Warren L, Beckman BC: *System for retrieving reusable software*, 1993, NASA Tech Briefs.
- Ward TP, Bracket WJ: *Object-oriented requirements definition and software design*, Boston, 1991, Boston University.
- Ward TP, Stephen JM: *Structured development for real-time systems*, vols I, II, III, Englewood Cliffs, NJ, 1985, Prentice-Hall.
- Warnier JD, Kenneth T: ORR, *structured systems development*, New York, 1977, Yourdon Press.

References

- Wasserman AI, Pircher PA, Muller RJ: *An object-oriented structured design method for code generation*, *SIGSOFT Software Engineering Notes* 14(1):32, 1989.
- Wilson W, Rosenberg L, Hyatt L: Automated analysis of requirement specifications, Fourteenth Annual Pacific Northwest Software Quality Conference, 1996.
- Wirfs-Brock R et al: *Designing object-oriented software*, Englewood Cliffs, NJ, 1990, Prentice Hall.
- Wirth N: Program development by stepwise refinement, *Communications of the ACM* 14(4):221, 1971.
- Wood W, Long G, Wood P, David P: *Classifying software design methods*, Technical Report, CMU/SEI-89-TR-25.

List of Figures

Chapter 1: Information Technology Project Initiation

Figure 1–1: Various phases of the IT project

Figure 1–2: Cost of hardware and software trends with time

Figure 1–3: Context diagram

Figure 1–4: Relationship among users, customers, and stakeholders

Figure 1–5: Major phases of an IT project

Figure 1–6: Generation air–ticket relationship among customers, users, stakeholders, and management

Chapter 2: Guidelines for Successful Project Planning

Figure 2–1: Modeling and simulation process

Figure 2–2: Sample WBS model

Figure 2–3: Sample master schedule

Figure 2–4: Budget model

Figure 2–5: Functional organizational model

Figure 2–6: Pure organizational model

Figure 2–7: Matrix organizational model

Figure 2–8: Hybrid organizational model

Figure 2–9: Staffing concept for a typical IT project

Figure 2–10: Generating ticket systems prototype model

Figure 2–11: WBS, manpower, and budget allocation

Chapter 3: Project Estimating Techniques and Tools

Figure 3–1 A: Symbols used in the Gantt chart;

Figure 3–1 B: Gantt chart showing project planning versus actual activity

Figure 3–2: A typical PERT network showing activities that must be performed and their sequence. Note that the dummy activity has no duration.

Figure 3–3: A, Activity; B, Event; C, Dummy; D, Restrictions

Figure 3–4: A, Another example of restrictions; B, Merge; C, Burst

Figure 3–5: A PERT network

Figure 3–6: Project activities

Figure 3–7: Project activities CPM

Figure 3–8: Project activities and Gantt chart

Chapter 4: Project Management Quality Control Mechanism

Figure 4–1: Capability maturity model levels

Figure 4–2: P–CMM model

Chapter 5: IT Project Risk Management

Figure 5–1: Safety risk constraints

Figure 5–2: Risk management process

Figure 5–3: Risk management plan activities

Figure 5–4 A: A well–managed IT project

Figure 5–4 B: A poorly managed IT project

Figure 5–5: Risk–monitoring process

Chapter 7: System Measurement Techniques

Figure 7–1: Quantitative project measurement process

Figure 7–2: Project measurement process

Figure 7–3: Sample cost, schedule, and project progress

Figure 7–4 A: Sample cost, schedule, and project progress

Figure 7–4 B: Sample funding and personnel resources

Figure 7–5 A: Bathtub curve (hardware defect);

Figure 7–5 B: Software defect

Figure 7–6: Tailoring process

Chapter 8: Commercial Items

Figure 8–1 A: Project performance efficiency

Figure 8–1 B: Tasks completed

Figure 8–1 C: Defects by phases

Chapter 9: Customer, Customer, and Customer

Figure 9–1: Knowing your customers parameters

Figure 9–2: Customer expectations

Figure 9–3: Customer acceptance scenario

Chapter 11: Internet Applications

Figure 11–1: WAP server

Chapter 12: Distributed Object Technology

Figure 12–1: A sample of distributed object environment

Figure 12–2: An overview of CORBA

Figure 12–3: DCOM client server application overview

Figure 12–4: Logical flow with distributed object technology

Figure 12–5: Before information access

Figure 12–6: After information access

Chapter 13: Distributed Objects

Figure 13–1: Window flow diagram

Figure 13–2: Client–server application

Figure 13–3: Various phases

Figure 13–4: Class interaction diagram

Figure 13–5: Customer contact window

Figure 13–6: Flight information

Chapter 14: Wireless Practical Case Study

Figure 14–1: How WAP works

Figure 14–2: Wireless tasks

Chapter 15: System Requirements

Figure 15–1: IT system requirements determination process

Figure 15–2: IT system requirement specification

Chapter 16: Managing System Requirements

Figure 16–1: Identification of defects in system requirements

Figure 16–2: Types of system requirement errors

Figure 16–3: Caliber–RM user interface screen

Figure 16–4: Caliber–RM requirement traceability screen

Figure 16–5: Caliber–RM project requirement baseline

Figure 16–6: Caliber–RM requirement grid screen

Chapter 17: System Design Process

Figure 17–1: IT system design process

Figure 17–2: A system architecture

Figure 17–3: A sample of a generic system architecture model

Figure 17–4: Financial accounting domain and subdomains

Chapter 18: Software Requirements

Figure 18–1: IT system design process

Figure 18–2: A sample of the incremental model

Chapter 19: Software Design Process

Figure 18–3: A sample of the waterfall model

Figure 18–4: A sample of the rapid prototyping model

Figure 18–5: A sample of the spiral life–cycle model

Figure 18–6: A sample of the domain prototype model

Figure 18–7: A sample context diagram

Figure 18–8: A sample data flow diagram. That data flow diagram consists of rectangles that are the source or destination of data outside the system. (Customarily, rectangles are not shown in the data flow diagrams because they are already present in the context diagram.) A data flow symbol (arrow) represents a path where data moves into, around, and out of the system. A process symbol (circle) represents a function of the system that logically transforms data. A data store is a symbol (open–ended rectangle) that shows a place in the system where data is stored.

DF, Data flow; P, process (data process); ST, store (data store); DS, data source; TM, terminator (information destination).

Figure 18–9: A context diagram

Figure 18–10: A sample of level 1 partitioning

Figure 18–11: A sample of level 2 partitioning

Chapter 19: Software Design Process

Figure 19–1: IT software design process

Figure 19–2: An OAM diagram

Figure 19–3: An OIM diagram

Figure 19–4: OIM table

Figure 19–5: OIM table and object relationship

Figure 19–6: OIM notations

Figure 19–7: Supertype and/or subtype constructs

Figure 19–8: Correlation symbol

Figure 19–9: Object lifetime diagram

Figure 19–10: OBM sample

Figure 19–11: Software–structured design

Chapter 20: Software Implementation Process

Figure 20–1: IT software implementation process

Figure 20–2: Integration of software systems with reuse assets

Figure 20–3: A sample solutions

Figure 20–4: Functional model (context diagram)

Figure 20–5: Functional model (data flow diagram)

Figure 20–6: STD sample solution

Chapter 21: System Integration and Evaluation

Figure 21–1: IT system integration process

Figure 21–2: A well–managed IT project

Figure 21–3: A poorly managed IT project

Chapter 22: Practical Case Study

Figure 22–1: Organizational chart

Figure 22–2: IT project schedule

Figure 22–3: IT project costs

List of Tables

Chapter 1: Information Technology Project Initiation

Table 1–1: Allocation Of It Project Resources

Chapter 2: Guidelines for Successful Project Planning

Table 2–1: Criteria For Selecting An Organization Model

Chapter 3: Project Estimating Techniques and Tools

Table 3–1: Baseline Plan Work Units

Table 3–2: Schedule Variance Work Units

Table 3–3: Cost Variance Work Units

Table 3–4: Spend Comparison Approach Work Units

Table 3–5: Case Study Date

Chapter 5: IT Project Risk Management

Table 5–1: Risk Analysis Criteria

Chapter 9: Customer, Customer, and Customer

Table 9–1: Customer Confidence Level

Chapter 13: Distributed Objects

Use Case Design Information

List of Boxes

Chapter 1: Information Technology Project Initiation

Box 1–1: List of Support Personnel

Chapter 6: Commercial Best Practices Standards

Box 6–1: Characteristics of a Good System Development Standard

Box 6–2: List of Data Item Descriptions

Chapter 9: Customer, Customer, and Customer

Box 9–1: Features to Help the Manager Understand the Customer

Box 9–2: Customer Expectation Factors

Chapter 15: System Requirements

Box 15–1: SysRS Outline