ISABEL EVANS

ACHIEVING SOFTWARE QUALITY THROUGH TEAMWORK

Achieving Software Quality through Teamwork

For a listing of recent titles in the *Artech House Computing Library,*
turn to the back of this book.

Achieving Software Quality through Teamwork

Isabel Evans

Artech House
Boston • London
www.artechhouse.com

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the U.S. Library of Congress.

British Library Cataloguing in Publication Data
Evans, Isabel
Achieving software quality through teamwork.—(Artech House computing library)
1. Computer software—Quality control 2. Computer software—Development—Management
3. Teams in the workplace
I. Title
005.1'0684

ISBN 1-58053-662-X
Cover design by Yekaterina Ratner

685 Canton Street
Norwood, MA 02062

International Standard Book Number: 1-58053-662-X

10 9 8 7 6 5 4 3 2 1

For my brother, James, statistician, rock climber, mountaineer, 1958–2003.
You encouraged me to write this book.

Contents

Foreword

If you are in IT, you probably think of yourself as a technical person. Often the emphasis is very much on the "technical" rather than the "person." Yet, as Isabel points out, the majority of problems in IT are due to people problems, not technical ones. Yes, producing quality software is a technical activity, but software is produced by people, complete with talents and abilities, but also personalities, idiosyncracies, foibles, and emotions, and these people produce IT systems in teams, where the roles and perspectives of each team can differ significantly, especially about what constitutes quality.

If every person has a different view of what quality is, and the people involved don't communicate well, is it any wonder that major problems arise? What is the solution? Is it better processes? That has been tried. Is it a maturity model? That has been tried. Yet problems still abound, and people are still surprised by the fact that there are still problems with people!

Isabel sprinkles this book with numerous "overheard conversations" within IT organizations. Why do customers find developers arrogant? Why do developers lack appreciation of business issues? Why do support staff feel left out? Why do managers not appreciate the value of testing? I frequently find myself in conversation with IT people, particularly testers and test managers, and I hear them saying many of the things that Isabel describes. This book not only explains what is going on, but also shows how things can be improved, bringing the ideas to life through Isabel's rich source of anecdotes. Reading this book will help you understand the viewpoint of the very people you often complain about!

The use of the EFQM Excellence Model as a framework to structure the book gives a solid foundation for comparing different models of quality and shows where the different roles of the people involved fit together in a software development project and the life of that system. From a technical perspective, the book gives useful guidance on achieving quality; Appendix B includes a summary of quality documents.

The project advice contained in the start-up section in Chapter 9 alone could be worth the price of this book; it could save you a lot of time, money, and mistakes and is an aspect frequently overlooked and underemphasized. Appendix A is also a useful handbook on its own—a concise summary of

people- and teamwork-related techniques and methods and where to go for further information.

This book will increase your understanding of the people with whom you work. It may cause a wry smile or two as you recognize the behavior described "to a T" of a colleague. Then you may find yourself thinking, "So that's what they're thinking, that's why they do that; I never realized." Perhaps you will even recognize yourself and realize why other people don't seem to understand your view of quality.

This book is a rare breed. It explains why people issues are important, yet it does so in a way that will appeal to technical people.

Dorothy Graham
Senior Consultant
Grove Consultants
Macclesfield, United Kingdom
May 2004

Preface

The cost of failed IT projects in the United States was recently estimated at $84 billion in just 1 year [1], so software quality matters more now than it ever has, and it matters to you and me because we use that software. For all of us, our reliance on software is increasing year by year whether we realize it or not. More of us are using software for more tasks than ever before, in information technology (IT), information systems (IS) for businesses, embedded systems in consumer goods such as phones, and, of course, across the Internet [2]. The risks associated with software failure have increased with the use of software; these include greater exposure for organizations when software fails or is unsatisfactory, and greater disappointment or loss for individuals when they are let down by software.

Meanwhile, pressures on modern organizations, including businesses, have increased in recent years. Pressures on organizations—the importance of time to market, cost reduction, value for money, increased expectation and knowledge of customers, global communications, constant change, and the need to find new markets—become pressures on software teams to produce more software, more quickly, with increased expectations of what that software can deliver as benefits.

Why is IT so often disappointing? Why isn't software built correctly? One reason is quite simple: IT systems are built by people, and people make mistakes. This is true for any human activity, but in IT we have a number of exacerbating circumstances:

- IT systems are often built by teams of people other than those who will use them. In these circumstances, any poor communication between people and teams increases the chances of mistakes.
- IT is relatively new, and we are working on a continuous learning curve. Both the suppliers and users of software rarely have time to consolidate knowledge before they face yet more change.
- IT departments are notorious for their failure to align the software they produce with the culture, processes, or objectives of the business for which the software is intended.

- IT systems are complex, and are becoming increasingly so in themselves and in their intercommunications with other systems.

Only one of these (the last) is a technical problem, yet most of the emphasis for IT groups seeking improvement is on technical processes. All the other points in the list have to do with people, their ability to communicate well and understand each other, and their ability to learn from each other and from experience.

We need a framework for IT projects that addresses the issues of software quality through an emphasis on teamwork and communication set in a framework aligned with the customers, their organizations, and their goals. To help achieve this goal, *Achieving Software Quality through Teamwork* answers the following questions:

- Who should be involved in the development and deployment of software? People are going to work in teams to provide the software, so we need the right teams.
- What are the differences and similarities between these people, especially in their assumptions about quality? To achieve quality, the team needs to agree on what quality is.
- What are the ways of understanding communication preferences and how conflict can arise from differences between these preferences? Teamwork requires mutual understanding and tolerance in communication.
- How can that understanding be improved? By providing opportunities for communication within and around the IT and organizational processes, teamwork and communication are encouraged so that quality is achieved.
- How can IT suppliers understand the goals of their client organizations, whether nonprofit or commercial businesses, and how they measure success? IT suppliers must have an understanding of the quality framework used by their customers in order to produce quality software. IT teamwork means including the customers.

This book was written in response to a number of people with whom I have worked over the years. It is for you if you, like they, have ever said things like:

- We've improved the processes, so why are the customers still unhappy?
- I just can't talk to the people in the business units (or IT, or management, and so forth). How can we understand each other?
- Why do they keep sending e-mails to me when I'd rather talk face-to-face?
- How can I provide quality if the managers just talk about costs?

- How can I align my goals with what the customer really needs?
- Why do the IT people always deliver the wrong thing?
- What do the managers really do?
- I have just started as a team leader. What do I need to think about?
- Who needs to be involved at this stage?
- Why don't the people in my team get along?
- Why do we always end up having an argument?
- We could get on with this if people stopped arguing!
- We've done some process improvements, and we've still got problems.
- I can't stand all this touchy-feely people stuff. Can't you just give me a process?
- Why do the IT people always cringe when I want to do a team-building exercise?

I hope you enjoy yourself!

As I wrote, I imagined you reading this book. One colleague who reviewed some chapters said to me, "I want to imagine that we are sitting by the fire with a glass of wine, sharing ideas and experiences," and that is what I wanted the atmosphere of the book to be like. I have used experience-based anecdotes rather than scientifically gathered evidence. The stories I tell include lessons I have learned (and am still learning!) from my own mistakes, situations I have observed, and anecdotes from colleagues and clients, and I hope that when you read them you will respond with "Oh, yes! That reminds me of when…."

So enjoy this book; feel free to browse and dip into it as well as read it through. A key message from it is that there is more than one way of thinking, and we are sensible to acknowledge them all, however strange they seem to us. So bring your own ideas to the book and read interactively!

This is a huge subject

This book is for you if you need an overview of a huge subject in one book, and the chance to find out more about the details that particularly interest you. I have covered the whole life of a software system, together with describing all the people involved, so this is a sketchbook of what you need to consider, with extensive references to further information.

As you are busy, when you want further information you will need to get to it quickly and easily. To help you, wherever possible I have given several references whenever possible to newer publications, including Web sites, short books or papers, or books that I have been able to locate in a library rather than have to order, so that you can get the next level of detail quickly

and easily. I have also provided original book or paper citations, either in the references or in the selected bibliography in each chapter, so that if you just need a little more information you can get it easily, but you can also find more depth when you need it.

In Appendix A, I have given some suggested Internet search words for each technique that I describe, as some of the references, especially for Web sites, may change. I have also provided a list of useful organizations from which you will be able to find more information.

Remember, this book is an overview and as a result there are many references, techniques, standards, methods, and frameworks that I have not covered, but that should not inhibit you from considering them for your own situation. If you prefer a different method at a particular point, simply incorporate it into your recipe for success.

Finding your way around this book

Three of the chapters in the book provide overviews of ideas presented in the chapters that follow them:

- Chapter 1 provides an overview of quality concepts used throughout the book.
- Chapter 2 provides an overview of the groups discussed in Chapters 3 to 7.
- Chapter 8 provides an overview of the software life span described in Chapters 9 to 12.

Chapter 1 sets out some ideas and definitions that are used throughout the book. Read this chapter first, as the ideas introduced are used throughout the book. I describe five definitions of quality and a number of quality or excellence frameworks that provide the basis for the discussion in the rest of the book. Each definition of quality provides a different viewpoint about quality—what it is and how we might measure it. Some of the frameworks are organizational—the business will use them to set and check its direction. Others are IT-specific. I show how the organizational and IT standards can fit together.

Chapter 2 provides an overview of the groups of people who are involved in software, whether as producers or users. I have divided people into five groups: customers, managers, builders, measurers, and supporters. I am not suggesting that an individual is assigned a role within a group, or always stays within one group. Instead, I show how the five groups fit with particular quality viewpoints, and that many people move between groups.

Chapters 3 to 7 each describe one of the groups in detail. Most readers will find that they fit most closely into one of the groups, but spend some time in the other groups. Read these chapters to get an overview of what each group does and what its viewpoints are, so that you understand each other better. Each of these chapters provides an overview of that group,

based on the organizational framework and the quality viewpoints described in Chapter 1:

- Chapter 3 describes the customers and users of software systems.
- Chapter 4 describes the people who manage software projects and services.
- Chapter 5 describes people who build products, not just the developers but also technical authors, training providers, business analysts, and designers.
- Chapter 6 describes people who measure the quality of products and processes: the testers, inspectors, document reviewers, and auditors.
- Chapter 7 describes the people who support the software during its development and deployment, by providing infrastructure for the software and tools used to build and test it, as well as supporting the people who build, test, deliver, service, and use the system.

Chapter 8 gives an overview of the life of a piece of software or a system from the moment it is conceived as an idea, through the software development life cycle, as it is installed or delivered, throughout its deployment as a live or production system, its maintenance and updating, and, finally, its decommissioning. It introduces Chapters 9 to 12, in which I discuss how communication needs to be considered and improved by all the groups throughout the life of a software system. I have only described the techniques applied by particular groups when they have an effect on communication and teamwork; the chapters are not intended to explain everything about software projects but to highlight some aids to mutual understanding. For example, quality gates, or entry and exit criteria, between stages in a project are important for process definition, and reviews are important for identifying defects in products but they are both also important communication points between the groups described in Chapters 2 to 7.

- Chapter 9 discusses what happens before we start to build software—we realize we have a problem to solve or an opportunity to grab, and we must decide whether we need software or something else to solve our problem. It covers problem and solution analysis, risk analysis, and setting the contract for the software development life cycle, describing how to improve communication and understanding in order to launch the right project.
- Chapter 10 describes the software development life cycle and provides an overview of some models for software development, comparing and contrasting the models' effect on effective communication and teamwork between groups.
- Chapter 11 describes the point of implementation of the software, types of delivery, and what information is needed by different groups at this point.

- Chapter 12 describes the life of software once it is in use, and discusses the importance of continued communication for evaluation of the previous stages and for maintenance and optimization of the software.

Throughout these chapters I refer to a number of techniques for understanding other people's communication styles. In Appendix A, I provide a summary of these techniques and sources for more detailed information. In Appendix B, I provide more information about tailoring standards and documents for a project based on published national and international standards.

References

[1] Smith, K., "The Software Industry's Bug Problem," *Quality Digest*, reproduced on http://www.qualitydigest.com, April 2003.

[2] Sol, E. -J., "The Embedded Internet—Towards 100 Billion Devices," *EuroSTAR Conference*, Stockholm, Sweden, 2001.

Acknowledgments

Naturally, a book like this is only built on other people's work and efforts, and you will see from the references that many practitioners and authors have supported me by being earlier writers in the same areas of thought. This book weaves together threads that others have spun. I thank them all for their inspiration over the years.

Many people helped me while I was writing this book, by their encouragement, discussing ideas, providing support, and reviewing material. Colleagues and clients have contributed with enormous generosity, discussing and challenging ideas, commenting and reviewing material and drafts, suggesting additional references, and providing valuable insights into the subject. They encouraged me to continue; their comments, stories, and ideas have improved the book immeasurably. I wish to thank, among others, Rick Craig, Dorothy Graham, Frank Johnstone, Mike Bowdon, Stuart Reid , Paul Gerrard, Richard Warden, Tom Gilb, Julian Harty, David Hayman, Mike Holcombe, Brenda Hubbard, John Smith, Norman Hughes, Kai Gilb, Jane Jeffs, Simon Mitchley, Fiona Powell, Lloyd Roden, Mike Smith, Jayne Weaver, Graham Thomas, Geoff Thompson, Neil Thompson, Erik van Veenendaal, John Watkins, Steve Allott, Jayne Weaver, Clive Bates, Mark Fewster, Pat Myles, and Ian Bennett. I could not have completed this without the help of Barbara Eastman, who encouraged me, proofread drafts, and pursued permissions. Richard Delingpole's graphic design expertise turned my rough ideas into elegant figures. Tiina Ruonamaa, Tim Pitts, and the team of editors at Artech House supported me throughout the project and kept me on track.

It is an honor for me that Dorothy Graham has written the foreword to this book. Thank you, Dorothy, for your help, encouragement, and friendship.

My family has supported my writing during a bad year for us all—thank you. Finally, my partner, Dave, has been an unfailing support throughout—thank you, Dave, for everything you have done.

My thanks to all of you for your help; the book would not have been possible without you. As I am human, there will be mistakes in the book; the mistakes, of course, are my own.

CHAPTER

1

Contents

Software Quality Matters

In this chapter I shall:

- Demonstrate that there is no universal definition of quality: it varies with people and situation;
- Offer several definitions of quality;
- Show why it is necessary for all stakeholders in the software to agree on what they mean by "quality";
- Introduce some models for managing quality;
- Show how you can integrate these models.

The developers are always so enthusiastic about the wonderful new software when they hand it over for me to develop the training. Then the "buts" start … they tell me that when I demonstrate the software I need to keep away from this transaction because of the outstanding defects, and to watch out for the finance director who's still sore about the budget overrun. They tell me the interface the users wanted wasn't feasible and preparation of user manuals has been de-scoped to a postlaunch activity. Once, on the morning of the first course, the company announced that the new software would "enable moving 40% of jobs overseas." "Negative trainees" would be an understatement!

—Trainer pointing out some forgotten aspects of quality

1.1 Defining software quality

Let us start looking at quality by examining the story above. What do the trainer's frustrations reveal about our views of quality?

1. The delivered software includes *other products and services* as well as code; the people buying and using software do not just need the code, they also need services and products such as training, user guides, and support.

2. *Quality cannot be defined by technical excellence alone*—it also includes human factors such as communication and motivation, as well as value for money. The customer must be able to afford the products and services, and enjoy the experience of dealing with the supplier as well as using the products.
3. Different people will hold various views on whether quality is delivered *to the customer*. The developers produced a fine product, with lots of wonderful new features, so they were rightly enthusiastic, but the product had flaws, cost too much, did not meet the customer's needs, and made the lives of their colleagues in training much harder. Also, the people losing their jobs would take a decidedly different view about whether a quality solution had been delivered.

We will find throughout this book that different people give different definitions if we ask them what they mean by quality. When I ask people working with software as users or on software development projects what quality means, all sorts of ideas emerge. Some might mention cost, time, scope, specification, value, or standardization, whereas others talk about perfection, expectations, relationships, feelings, and emotions. Some of this can be accomplished by delivering code, but much is achieved from other software products and services such as documentation and training. Some can only be realized through less tangible things such as relationships and expectations.

Why is this important? Because it causes difficulties when people involved in a project have different views of quality. If people do not have shared goals and aspirations, their view of quality is affected. It means that, if they do not communicate their differences and negotiate a common goal, they may work against each other, while thinking they are all pulling together. There will be confusion about whether a delivered software product provides quality. So many systems are delivered that are not used, or not liked, or cannot be supported by training, documentation, and help provisioning. Yet the IT project teams are either unaware of, or surprised by, the reaction to what they have perceived as a successful project. We all believe we have done a good job and are mortified if our work is not well received! We commonly find a different view of whether a project has delivered a "quality product" if we talk to the customers and to the project team. Furthermore, we frequently find differences of opinion if we talk to the project manager, the developers, the testers, the trainers, and other groups within the project team. Each person may seem to hold a different view of the quality of the product and service, and, indeed, a different underlying assumption of how to measure quality. So I will start by defining quality from a variety of viewpoints.

Five distinct definitions for quality can be recognized.[1] The definitions I will use are from [1]. They are the *product*-based, *manufacturing*-based, *user*-based, *value*-based, and *transcendent* definitions.

1. The definitions of quality in this chapter and throughout the book are based on Chapter 1 of [1], reused by permission of UTN Publishing. These definitions are based on work by [2] and were adapted by [3].

Two definitions of quality favored by IT people are the product-based and the manufacturing-based definitions. In projects, we favor definitions that allow us to measure progress and success in delivery. We want to fix quality to something that is deliverable and measurable.

- In the product-based definition, quality is based on a well-defined set of software quality attributes that must be measured in an objective and quantitative way. We can derive acceptance criteria to objectively assess the quality of the delivered product. *Example: Standards such as ISO 9126* [4] *define attributes such as reliability, usability, security, and functionality, together with measures for them. "The software is 98% reliable when running continuously over a 7-day period. Recovery time is less than 1 minute at each failure."*
- The manufacturing-based definition focuses on the manufacture of software products, that is, their specification, design, and construction. Quality depends on the extent to which requirements have been implemented in conformance with the original requirements. We measure faults and failures in products. Success is measured as our ability to follow a process and deliver products against agreed specifications. We *will* verify (is the system correct to specification?) but if we do not take account of the *user-based* definition of quality (see below), we *may forget* to validate (is this the right specification?). *Example: Repeatable, auditable process with delivery that conforms to specification. "The software was built to specification, and there are a low number of defects."*

There are two other definitions of quality that reflect the views of the software user and purchaser. These perspectives are about supporting the needs of the organization and its stakeholders, within the organization's constraints. Because the pressures on an organization change over time, what constitutes "quality" may change over time to match. Sometimes the changes will be tactical—*"We must cut costs this quarter!"*—and sometimes they may be strategic—*"We want to be market leaders!"*—and these may conflict.

- The user-based definition says that quality is fitness for use. Software quality should be determined by the user(s) of a product in a specific business situation. Different business characteristics require different types of software products; not only to do different things, but also to cater to how different people want to carry out their tasks. This can be subjective and cannot be determined only on the basis of quantitative metrics. It is the user-based definition that encourages us to validate *as well as* to verify the system. *Example: Fit for purpose. "I can do my work efficiently and effectively when I use this software."*
- The value-based definition is focused on things that impact on the running of the business as a whole. Software quality should always be determined by means of a decision process on trade-offs between time, effort, and cost aspects. This is done by communicating with all parties involved, for example, sponsors, customers, developers, and

producers. *Example: Return on Investment (ROI). "If we release the software now, we will spend $250,000 extra on support in the first month. If we are a month late, it will cost the organization $1 million in fines and lost business. Should we release or do more testing?"*

Finally, we must acknowledge that we all know quality when we see it; our knowledge is based on our experiences, taste, affections, loyalties, and emotions. Unfortunately, this means different people will have different reactions to a product: The transcendent definition states that quality can be recognized easily depending on the perceptions and the affective feelings of an individual toward a type of software product. This means that we consider someone's emotional response to a product or service. *Did they enjoy it? Did they like the people they met? Are they happy?* This is not easily measurable, but understanding this aspect of quality can be a first step toward the explicit definition and measurement of quality. *Example: Brand loyalty based on affection. "I like using this software—it appeals to me."*

In a single project, we may have several definitions of quality in use, perhaps inadvertently and unacknowledged by all the people in the project. It is important to realize that there is no "right" definition of quality. How we define "quality" for a particular product, service, or project is situational. We say, "*It depends on*…." Contrast the following:

- *Air traffic control system:* We are considering using the product/manufacturing definitions because we need to ensure technical excellence above all else.
- *Package to improve usability of Web pages for the visually impaired:* We are considering using the user-based definition because we need to ensure it is fit for the purposes of this group.
- *Software to launch an innovative new product and achieve "first mover" advantage:* We are considering using a value-based definition because if we spend more to get a better product we may miss the market.

For most commercial or custom-made software, the customer is best served by balancing the definitions. In our particular project, we should ask ourselves: What is the greatest number or level of attributes (product-based) that we can deliver to support the users' tasks (user-based) while giving the best cost–benefit ratio (value-based) while following repeatable/quality-assured processes within a managed project (manufacturing-based)?

As a colleague remarked to me, "Compromise and a balance between the quality definitions is essential" (Frank Johnstone, personal communication, April 18, 2003). We *need* to define quality and to understand which definitions people buying, using, developing, testing, and supporting software use. This prevents the conflicts between stakeholders and enables us to understand *why* we are developing the software. In Chapters 2 to 7, we will explore the different groups and which quality definitions they favor, and then, in Chapters 8 to 12, we will look at how the different groups contribute

throughout the life of software and systems, and the benefits that each quality view brings.

Whichever definition(s) of quality we use—that of software users or suppliers—we all want to try to avoid mistakes. One way to try and do this is to adopt strong processes within a quality management framework. In *quality management*, our concern is to decide which processes to use and to adapt those processes appropriately for a project. During the project, we will carry out *quality assurance* activities to check that the chosen process has been followed and is suitable. *Quality control* processes will check the products for defects. The processes sit within an organizational culture and framework of management systems. Some quality management models are entirely *process driven*; our task as people operating within the processes is to follow the processes as defined. If we are lucky, we might be asked to suggest improvements to the processes. Other models, specifically those described as excellence models to differentiate them from process-driven quality models, are more focused on *people and their capabilities and needs*; here the activities of the project are focused on the ability of people to deliver services and products that satisfy the customers. Some models have a greater emphasis on improvement cycles than others; the *Deming cycle*, for example, proposed by W. Edwards Deming [5], has four stages, sometimes called the "Plan, Do, Check, and Act" cycle, and sometimes the "Plan, Do, Review, Improve" cycle. Here, we plan what to do and we do it. Then we review what we did. Was it successful? Did it go as planned? What should we improve? We then put improvements in place, and plan the next cycle of activities. We will see through this book that this Deming improvement cycle works on a large scale, for example, when looking at the excellence framework for an organization or for a project, but is particularly effective for making incremental improvements as we do work, whether as a team or individually.

We need to look at these models and understand the advantages and disadvantages of each one.

1.2 Fundamental concepts of excellence

To compare models, I shall use the Fundamental Concepts of Excellence set out by the European Foundation for Quality Management (EFQM) [6]. These concepts are used in a number of models, including the EFQM Excellence Model, which is used by more than 20 member countries in the European Union. Similar organizational models based on these concepts have been developed in other countries. For example, Puay et al. [7] compare nine schemes, including the Malcolm Baldrige model [8] and the EFQM Excellence Model [6], in different countries (three European, two North American, three Asian-Pacific, and one South American) against nine criteria: leadership, impact on society, resource management, strategy and policy, human resource management, process quality, results, customer management and satisfaction, and supplier/partner management and performance. It is these criteria that are reflected in the Fundamental Concepts of Excellence.

Other organizational quality and excellence initiatives, such as Six Sigma and the Balanced Scorecard, also provide a way of discussing the goals of an organization, deciding how to achieve those goals, and measuring whether they have been achieved.

In this book, I focus on the EFQM Excellence Model, but whichever model your organization uses, whether Baldrige or one of the others, you should be able to map your model onto the ideas in this book. This is because, although the different models use slightly different words and place a different emphasis on the different criteria, the fundamentals of running a successful organization apply, worldwide. For example, I will link the EFQM Excellence Model to a number of other standards, specifically for use in IT. In the same way, in [9] the authors have linked Six Sigma to IT frameworks.

The Fundamental Concepts of Excellence of the EFQM Excellence Model [6] are:

- *Results orientation*: "Excellence is dependent upon balancing and satisfying the needs of all relevant stakeholders (this includes the people employed, customers, suppliers, and society in general, as well as those with financial interests in the organization)."
- *Customer focus:* "The customer is the final arbiter of product and service quality, and customer loyalty, retention, and market share gain are best optimized through a clear focus on the needs of current and potential customers."
- *Leadership and constancy of purpose:* "The behavior of an organization's leaders creates a clarity and unity of purpose within the organization and an environment in which the organization and its people can excel."
- *Management by processes and facts:* "Organizations perform more effectively when all inter-related activities are understood and systematically managed, and decisions concerning current operations are planned. Improvements are made using reliable information that includes stakeholder perceptions."
- *People development and involvement:* "The full potential of an organization's people is best released through shared values and a culture of trust and empowerment, which encourages the involvement of everyone."
- *Continuous learning, innovation, and improvement:* "Organizational performance is maximized when it is based on the management and sharing of knowledge within a culture of continuous learning, innovation, and improvement."
- *Partnership development.* "An organization works more effectively when it has mutually beneficial relationships, built on trust, sharing of knowledge, and integration, with its Partners."
- *Public responsibility.* "The long-term interest of the organization and its people are best served by adopting an ethical approach and exceeding the expectations and regulations of the community at large."

How can we encourage the Fundamental Concepts of Excellence and get the best from our software teams? If you look at the Fundamental Concepts of Excellence, you will see that process, which emphasizes manufacturing and product-based viewpoints, is only one part we need to consider. When we look at the concepts, we see they involve considering and working with different groups of people, and thinking about balancing the views of different stakeholders. To help us do this, I will use the concepts throughout the book, to encourage a teamwork approach and allow all five quality definitions. Some quality models define quality processes for an individual to use, some are for teams, and some for organizations. The coverage of the quality definitions and the emphasis on teamwork varies across the models. In this book, I will use a selection of the models. They are:

- A framework for organizational excellence, known as the EFQM Excellence Model [6] in Europe, which is similar to others, such as the Malcolm Baldrige model [8], used in the United States;
- Two process standards—ISO 9000:1994 and ISO 9000:2000 [10];
- A group of IT maturity models based around CMM® (Capability Maturity Model®) [11, 12] and two models for implementing CMM®: the Team Software Process (TSP) [13] and the Personal Software Process (PSP) [14].

1.3 EFQM Excellence Model

The European Foundation for Quality Management (EFQM) Excellence Model [6] is an organizational excellence model using a nonprescriptive framework. It is not specifically an IT framework. It may be used with organizations of any size and type and is intended for corporations, companies, or nonprofit organizations. Here I am using it to help discuss a "mini-organization": the software project. The EFQM Excellence Model provides a framework for excellence under nine criteria; five of these are "Enablers" for excellence and four are measures of the "Results." These are interlinked with a continuous improvement feedback loop known as RADAR (Results, Approach, Deployment, Assessment, and Review) (Figure 1.1).

1.3.1 Enablers

The Enablers are *Leadership, Policy and Strategy, People, Partnerships and Resources,* and *Processes*. The results are *Customer, People, Society,* and *Key Performance Results*. In the following descriptions, I will first give the description for an entire organization and then for a project as a "mini-organization."

1.3.1.1 Leadership

Excellence is led from the top. Leaders facilitate the achievement of the mission and vision, and develop values for success. Leaders are personally

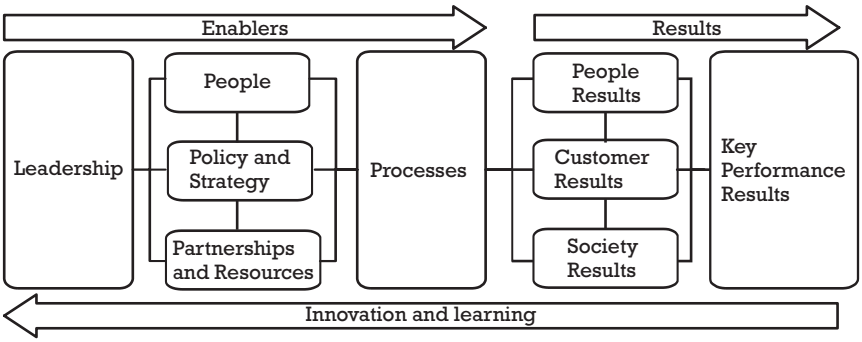


Figure 1.1 EFQM Excellence Model. (*From:* [6]. © 1999–2003 EFQM. Redrawn by permission of EFQM.)

involved in ensuring that the management system is developed and implemented.

Project managers provide leadership for their projects under the leadership of the project board and sponsor. In the mini-organization the project manager is the leader.

1.3.1.2 Policy and Strategy

The vision of leaders is implemented via policies and strategies. Leaders focus policy and strategy around the needs of the stakeholders and ensure they are reflected in policies, plans, objectives, targets, and processes.

The specific strategy for the project (including any quality strategy) is derived from the organization's overall policy and strategy. Additionally, it will show where changes have been made to reflect the needs of a particular project. The mini-organization requires its own strategies.

1.3.1.3 People

The organization manages, develops, and releases the knowledge and full potential of its people at individual, team-based, and organization-wide levels. The organization must consider how people are treated and valued.

In a project, for example, we need to consider not only who is working on the project and the skills they bring, but also how their skills and experience are enhanced during the project. During and after the project, people must feel that they and their contributions were valued.

1.3.1.4 Partnerships and Resources

The organization plans and manages its external partnerships and internal resources in order to support its policy and strategy and the effective operation of its processes. The organization considers partnerships with other organizations and how resources such as technology and information are managed. This divides into two distinct categories. The first includes the people outside our organization with whom we interact.

In the context of the mini-organization, I mean those external to our project. This might include how we liaise with suppliers and other projects. The second includes the (nonhuman) resources we require in order to complete our tasks, for example, management of the organization's information, IT infrastructure, and negotiation for scarce facilities such as development/test environments.

1.3.1.5 Processes

The organization designs, manages, and improves its processes in order to support its policy and strategy and fully satisfy, and generate increasing value for, its customers and other stakeholders. In a project, we need to consider which processes are appropriate. If we have experienced people and low-risk problems to solve, we might consider a lightweight, agile method for our IT project. If we have a high-risk project or inexperienced people, we may find that heavy-duty processes with an audit trail may give us more confidence. We select processes appropriate to our problem and our team.

1.3.2 Results

There are four sets of results measurements. The first three have a perception and a performance measure, whereas the fourth, key performance results, focuses on outcomes in relation to planned performance. As in the Enablers, I will show how the mini-organization of a project relates to the larger organization.

1.3.2.1 Customer Results

These measure whether the organization is meeting the needs of its external customers. Customer perception can be assessed through satisfaction surveys. Performance could be monitored by tracking the number of complaints and volume of repeat business. For a project team, or an IT team undertaking a series of projects, we might measure the perception of that team or project by its customers, the users of the software, and the managers of the purchasing organization.

1.3.2.2 People Results

Here, we measure what the organization is achieving for its people. The organization may survey employee perception and set performance targets for staff turnover and absence. Similarly, we could measure the motivation of the project team, people's attitude to working on the project, and factors such as illness and turnover.

1.3.2.3 Society Results

Society results measure what the organization is achieving in relation to local, national, and international society where appropriate. We can

measure how our organization is perceived in society, for example, through favorable press. A performance measure could be the attainment of awards, perhaps for corporate social responsibility.

Within a project, we might measure how the project measures against corporate targets for waste and energy management. We might also plan for, and measure use of, time for project members to carry out activities in the local community.

1.3.2.4 Key Performance Results

These results measure what the organization is achieving in relation to its planned performance, including financial measures such as return on investment, profit, and turnover, and nonfinancial outcomes such as market share and sales success rates. For the project, indicators of its success might include contribution to achieving increased business and decreasing costs, but we might also want to measure how well the project delivers against its planned budget.

1.3.3 Excellence, the EFQM Excellence Model, the Malcolm Baldrige model, and other related models

As we might expect, the EFQM Excellence Model meets the Fundamental Concepts of Excellence well. It is a European model, but is closely related to other models such as the U.S. model, the Malcolm Baldrige model [8]. The Baldrige model has the same aims and a very similar framework. In both, organizations score points against the enablers and the results to accumulate a total excellence score out of 1,000. Organizations compete against themselves—they seek to improve their score year by year. If organizations wish, they can compete against other organizations in an award scheme. The point of both models is to encourage the continuous improvement of organizations, rather than to achieve a specific level.

1.4 ISO 9000:1994 and ISO 9000:2000

ISO 9000:1994 [10] is a quality assurance standard for design and manufacturing processes. It is rigid in its definition and interpretation of quality. A large number of processes are defined and must be adhered to by means of audit trails and evidence, regardless of whether these processes give the best outcome for the customer or for the project team on a particular project. In ISO 9000:2000 [10], there are a smaller number of defined and documented processes and a greater emphasis both on people understanding the tasks that they have to perform and on customer satisfaction. ISO 9000:2000 includes continuous improvement and moves toward the EFQM framework. Although the ISO 9000 standards meet some of the Fundamental Concepts of Excellence, there are significant gaps, especially in ISO 9000:1994. However, the standards are useful in IT projects as a guide for auditability.

1.5 IT maturity models—CMM® and relations

The concept of organizational maturity and capability for an IT organization was developed at the Software Engineering Institute (SEI) [11] to define a better way of producing software. It regards software as an engineering discipline and groups organizations into five levels within the Capability Maturity Model® (CMM®):

- *Level 1—Initial:* Projects are ad hoc and chaotic. "Everyone has their own process."
- *Level 2—Repeatable:* Requirements are managed and projects are performed according to documented plans. "Every team has its own process; teams can repeat work."
- *Level 3—Defined:* Software engineering and management processes are stable and do not break down under stress. "Every team in the organization uses the same process; we can start to deal with change."
- *Level 4—Managed:* The organization manages its processes quantitatively and measures performance and quality across all projects. "The whole organization is measuring so we KNOW how we are doing."
- *Level 5—Optimizing:* Continual improvement and proactive defect resolution. "We can build on our knowledge to improve."

Progress through these levels is measured by key process indicators (KPIs). All the indicators at one level must be met before an organization can be considered to be at that level. The levels and KPIs are focused on the software development process and measurement of that process:

> The CMM® provides a staged approach to IT process improvement. The underlying premise is good common sense: the IT organization needs to walk before it runs. Sophisticated engineering and measurement processes cannot be sustained unless they are built upon a framework of strong basic management practices. Organizations that omit embedding Level 2 processes normally return to "ad hoc and chaotic" in periods of stress. (Frank Johnstone, personal communication, April 18, 2003)

CMM® covers software development, and considers testing as a part of this, with explicit requirements for testing included at Level 3 and above. To enhance this, number of testers started to develop related models specifically for testing. These include TMM® [15] and TPI® [16], among others [17]. These are all process models. They differ from the Fundamental Concepts of Excellence in that they generally focus on process and measurement of process to the exclusion of other issues, although, as the models develop, increasing heed is taken of the wider aspects of excellence. Difficulty with the implementation and use of CMM® gave rise to two further process models: the Team Software Process (TSP) and the Personal Software Process (PSP), which are described below. CMM® and its relations continue to develop. Recently the Software Engineering Institute has introduced the

CMMI® (CMM® Integration) and PCMM® (People CMM®) models, which attempt to widen the applicability of the CMM® concept to any engineering discipline and to cover management of people. For the latest news on CMM® and its relations, please visit the Software Engineering Institute Web site [11]. A paper comparing some test assessment and improvement processes was given by Stuart Reid at EuroSTAR 2003 [17].

1.6 Team Software Process and Personal Software Process

The Team Software Process (TSP) was developed at the Software Engineering Institute "to help integrated engineering teams more effectively develop software-intensive products. This process method addresses many of the current problems of developing software-intensive products and shows teams and their management explicitly how to address them" [11]. The TSP identifies that software projects fail because of teamwork problems and not because of technical issues. In [13] Watts Humphrey identifies ineffective leadership as a key problem for teams. The TSP requires the establishment of goals, the definition of team roles, the assessment of risks, and the production of a team plan. TSP permits whatever process structure makes the most business and technical sense. Teams are self-directed; in other words, they plan and track their own work. Managers operate by coaching and motivating the teams. In using the TSP, compliance to CMM® Level 5 is expected. We can see some common ground with the Fundamental Concepts of Excellence, particularly in the areas of leadership, people, process, improvement, and measurement. There are gaps in focus on the needs of the customer. In addition, the assumption of CMM® Level 5 means that the TSP is not so useful for organizations with less mature processes.

The Personal Software Process (PSP), developed by the Software Engineering Institute and based on CMM®, is a process definition for software engineers enabling them to plan and track work. PSP provides a framework of processes for the software engineer. It emphasizes the need for individual software engineers to receive intensive training before they use the processes. Good process is needed, but that will only work if people understand and are motivated to use the process. "Seventy percent of the cost of developing software is attributable to personnel costs; the skills, experience, and work habits of engineers largely determine the results of the software development process" [11]. This fits well with some aspects of the Fundamental Concepts of Excellence; it considers process and people's skills. However, the needs of the customer are not a focus for these models:

> The Personal Software Process helps individual engineers to improve their performance by bringing discipline to the way they develop software. ... It is not a matter of creativity versus discipline, but of bringing discipline to the work so that creativity can happen. … The PSP shows engineers how to manage the quality of their products and how to make commitments they can meet. It also provides them with the data to justify their plans. [11]

1.7 Bringing the models together

Having briefly outlined these models, we can see how they complement each other. CMM® uses a staged approach to IT process improvement, which prescribes processes that the organization is ready to receive. PSP and TSP acknowledge the importance of people and teamwork in implementing and using processes. ISO 9000 shows us how to develop auditable processes. It is possible to fit all our organizational standards into a framework like the EFQM Excellence Model, and the ethos of the awards scheme is to encourage organizations at a low level of maturity to take the first steps toward excellence by assessment and improvement. It is possible to self assess and work for initial improvements, but continuous improvement is encouraged by assessment and comparison with other organizations at a regional, national, and European level. However, it does not have an IT focus, so in Figure 1.2 I have overlaid onto the EFQM Excellence Model the methods, processes, and standards that have been mentioned so far. In Chapters 3–7, I will discuss how the EFQM Excellence Model can be used as a framework to help leadership, strategy, and policy aid individuals or teams in understanding their own objectives and how they fit with their organization.

Each model has gaps and most do not encourage all the five definitions of quality (see Table 1.1). Only the EFQM Excellence Model, with its measurements of perception as well as performance, acknowledges the *transcendent* view of quality. At present, only the EFQM Excellence Model, with its measures of key performance results, including financial results, acknowledges the *value-based* view of quality. PCMM® performance measurement is set against the organization's business objectives, but not yet, as far as I can ascertain, as a value-based quality. However, together the models have strength; from the ISO 9000 family we can use the idea of evidence and audit trails, from the CMM® family we can use the idea of developing maturity of process, and from the EFQM we can use the quality concepts of

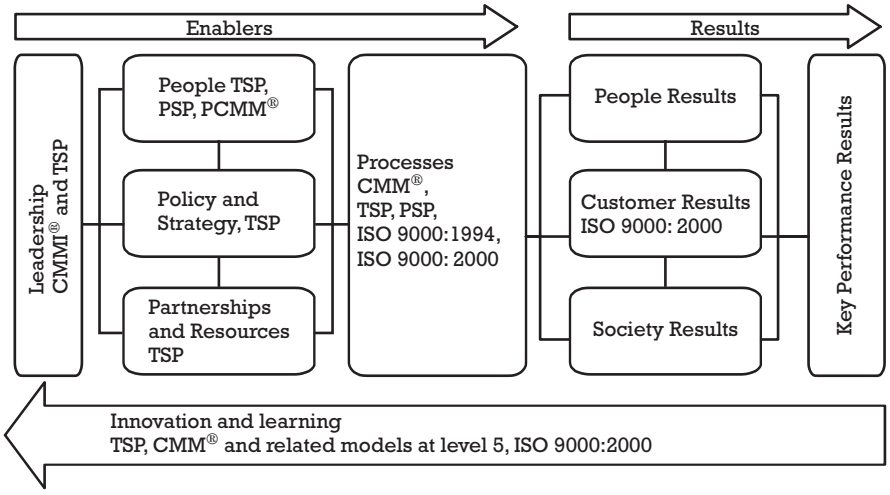


Figure 1.2 How the models fit in the EFQM Excellence Model. (*After:* [6].)

Table 1.1 Views of Quality Across the Models

	Model			
Quality View	*EFQM/Baldrige*	*CMM® and Relations*	*ISO 9000:1994*	*ISO 9000:2004*
ISO 9000:2000	✓			
Transcendent	✓			
User	✓			✓
Value	✓	(✓)		
Product	✓	✓		
Manufacturing	✓	✓	✓	✓

✓ is a primary quality view.
(✓) is a quality view that may be taken by some people in this group.

value and transcendent excellence. Organizations can benefit from any or all of these models. Meeting all the requirements of a model is rarely necessary or an end in itself. Organizations can select aspects of the models and choose from the techniques suggested, to meet their specific needs. This is an approach that I encourage throughout this book.

There are other standards that apply to software development, delivery, and support. It is important to realize that these standards will always sit within an organizational and cultural framework. Our choice of particular standards will reflect how we define quality, our industry/sector, our process maturity, and what is appropriate for our particular project. These other standards I will cover as needed in the rest of the book. I will note here three references that also place software standards in a framework. We have already mentioned a paper that sets software standards within Six Sigma [9]. In addition, the Software and Systems Quality Framework (SSQF), [18] mentions the EFQM Excellence Model and Baldrige model, Six Sigma, CMM®, and ISO 9000 as possible frameworks within which software standards might be adopted, but concentrates on ISO 9000 as the example framework. A useful paper concentrating on testing-related standards can be found on the Testing Standards Web site [19], although it does not mention the EFQM Excellence Model and Baldrige model.

Finally, the move toward integrating standards together is becoming increasingly important; organizations are moving toward *integrated management systems* that integrate quality, environmental, security, and financial management systems into one framework, and also include information management within and beyond the IT systems. Standards for IT work cannot stand alone; they must be part of the organization's integrated management system in order to align with the organization's aims. These will align with excellence frameworks, whether these are the EFQM Excellence Model, Baldrige model, or another framework. Frameworks for IT service management already align with the EFQM Excellence Model and the Baldrige model [20]. Whether providing new technology or exploiting existing technology, what is needed is for IT development and for project management to share that alignment. As organizations develop new excellence and

management frameworks to face their changing world, IT development and support standards will need to follow.

References

[1] Evans, I., "Testing Fundamentals," in *The Testing Practitioner,* E. van Veenendaal, (ed.), Den Bosch, the Netherlands: Uitgeverij Tutein Nolthenius, 2002, pp. 13–30.

[2] Garvin, D., "What Does Product Quality Really Mean?" *Sloan Management Review*, Vol. 26, No. 1, 1984.

[3] Trienekens, J., and E. van Veenendaal, *Software Quality from a Business Perspective*, Deventer, the Netherlands: Kluwer, 1997.

[4] International Standards Organization/International Electrotechnical Commission (ISO/IEC), DTR 9126 Software Engineering—Software Product Quality (Parts 1–4, 2000/2001).

[5] The W. Edwards Deming Institute, "Deming's Teachings," http://www.deming.org/theman/ articles/articles_gbnf04.html, November 2003.

[6] European Foundation for Quality Management, "EFQM Excellence Model" and "Fundamental Concepts of Excellence," http://www.efqm.org, August 2003.

[7] Puay, S. H., et al., "A Comparative Study of the National Quality Awards," *TQM Magazine*, Vol. 10, No. 1, pp. 30–39.

[8] Malcolm Baldrige model, http://www.quality.nist.gov/index.html, August 2003.

[9] Gack, G. A., and K. Robison, "Integrating Improvement Initiatives: Connecting Six Sigma for Software, CMMI®, Personal Software Process, and Team Software Process," *Software Quality Professional,* September 2003, pp. 5–13.

[10] International Standards Organization, ISO 9000:1994 and ISO 9000:2000 Quality Systems.

[11] Software Engineering Institute, "Capability Maturity Model®," http://www.sei.cmu.\edu, July 2003.

[12] Caputo, K., *CMM® Implementation Guide: Choreographing Software Process Improvement*, Reading, MA: Addison-Wesley, 1998.

[13] Humphrey, W., *Introduction to the Team Software Process,* Reading, MA: SEI, 2000.

[14] Humphrey, W., *Introduction to the Personal Software Process*, Reading, MA: SEI, 1997.

[15] van Veenendaal, E., and R. Swinkels, "Testing Maturity Model," in *The Testing Practitioner,* E. van Veenendaal, (ed.), Den Bosch, the Netherlands: Uitgeverij Tutein Nolthenius, 2002, pp. 289–300.

[16] Koomen, T., and M. Pol, *Test Process Improvement*, Reading, MA: Addison-Wesley, 1999.

[17] Reid, S. C., "Test Process Improvement—An Empirical Study," *EuroSTAR Conference paper*, Amsterdam, the Netherlands 2003.

[18] British Standards Institute, PD0026:2003, *Software and Systems Quality Framework—A Guide to the Use of ISO/IEC and Other Standards for Understanding*

Quality in Software and Systems, London, England: British Standards Institute, May 2003.

[19] Reid, S. C., "Software Testing Standards—Do They Know What They Are Talking About?" http://www.testingstandards.co.uk/publications.htm, August 2003.

[20] IT Infrastructure Library, *Best Practice for Service Delivery,* Norwich, England: Office of Government Commerce, 2002.

Selected bibliography

British Quality Foundation, *How to Use the Model*, London, England: British Quality Foundation, 2002.

British Quality Foundation, *The Model in Practice 2,* 2nd ed., London, England: British Quality Foundation, 2002.

Burnstein, I., T. Suwannasart, and C. R. Carlson, "Developing a Testing Maturity Model," *CrossTalk*, August/September 1996.

European Foundation for Quality Management and British Quality Foundation, *EFQM Excellence Model*, London, England: British Quality Foundation; Brussels, Belgium: European Foundation for Quality Management, 2002.

Handy, C., *Understanding Organizations*, New York: Penguin, 1993.

Hayes, L., "Hello Up There! Will the Sarbanes–Oxley Act Finally Catapult QA to the Boardroom?" Sticky Minds Web site, http://www.stickyminds.com/sitewide.asp?Function=FEATUREDCOLUMN&ObjectId=6544&ObjectType=ARTCOL&btntopic=artcol&tt=LIMITCAT_6544_**WHERE**&tth=H, August 2003.

Kaplan, R. S., and D. P. Norton, *The Balanced Scorecard*, Boston, MA: Harvard Business School Press, 1996.

Larson A., *Demystifying Six Sigma*, New York: AMACOM, 2003.

Mullins, L. J., *Management and Organisational Behaviour*, 5th ed., New York: Financial Times/Pitman, 1999.

Seddon, J., and Vanguard Consulting, "Lean Service: Systems Thinking for Service Organisations—The Business Excellence Model—Will It Deliver?" http://www.lean-service.com/6-3.asp, November 2003.

Sticky Minds Web site Round Table, facilitator Craig, R., "What Is Software Quality and How Do You Measure Its Value?" http://www.stickyminds.com/s.asp?F=S6540_ROUND_46, August 2003.

Woodruff, W. D., "Introduction of Test Process Improvement and the Impact on the Organisation," *Software Quality Professional*, September 2003, pp. 24–32.

CHAPTER

2

Defining the Software Team

Contents

In this chapter I shall:

- Describe the stakeholder groups that make up the software team;
- Discuss the mutual distrust between the team members and how it might be overcome;
- Identify which definitions of quality from Chapter 1 best fit to each group;
- Identify which of the quality models from Chapter 1 best fit the groups;
- List techniques to help improve communication between the groups.

It makes me wonder why senior managers don't knock a few heads together. After all, we are all fighting on the same side, or should be.

—Comment from a test consultant, noting the disagreements between members of the software teams

2.1 Teams in disunity

Some years ago, I was working with a software team to help them improve the quality of their delivered software. I met a succession of people—first the testers, then the developers, then the business analysts, and, finally, a group of project managers. What was astounding was that each group was convinced that they were the least respected group and that they were the group that cared most about quality. They all described themselves as being "on the bottom of the heap," and commented on the lack of support and the grief they received from the other three groups. They all pointed to the software user and to senior management as "awkward customers." What was going on?

Each person held strong views about their own contribution and that of other people to the project and to quality. Studies of the motivation of IT personnel and users of IT systems have shown that people believe that their own group enhances the project and other groups detract from it. Warden and Nicholson, in their 1996 survey of motivation in IT staff [1], remarked:

> IT is not a close-knit community of like-minded professionals. Many negative attributions are made about other groups lacking the motivation for quality. Senior managers are accused of paying lip-service to quality, while starving it of resources in pursuit of profit. Software developers are accused of focusing on technical excellence, completely disregarding customers' need for a quality product. Customers are accused of demanding levels of quality which they are not prepared to pay for. These are among the most common criticisms but there are many others. Each group within the profession makes negative attributions about other groups.

Table 2.1 shows that each group has a positive self-perception, but often feels negatively about other groups. Many of you will recognize these interactions and misunderstandings.

I have illustrated all the chapters in this book with anecdotes from my own experience, and I hope that you will add stories of your own. Here are

Table 2.1 Group Attitude Results

View	*How* *Developers*	*IT Infrastructure Staff*	*Software Maintainers*	*IT Managers*	*Quality Practitioners*	*Testers*	*Users*
Developers	We're OK.	Don't listen. Poor quality.	Will not document.	No business awareness or interpersonal skills.	Resist change. A law unto themselves.	Resent criticism.	Computers are difficult to use and unreliable.
IT infrastructure staff	Always complaining.	We're OK.	Should filter requests.	Better customer orientation.	Understand.	Understand.	
Software maintainers	Unpleasant job, low status.	Long delays	We're OK.	Long delays.	Resist change.	Resent criticism.	
IT managers	Want delivery too soon to get quality.	Should force developers to attend to users.	Want delivery too soon to get quality.	We're OK.	No interest, no support. Don't understand.	Don't know the cost of letting bugs through.	
Quality practitioners	Waste time. Unrealistic. Don't understand.	Understand	Waste time. Unrealistic. Don't understand.	A luxury. Only if customers demand it.	We're OK.	Understand.	
Testers	Unnecessary —we do that!	Understand	Unnecessary —we do that!	Little skill needed.	Understand.	We're OK.	
Users	Don't know what they want.	Complain to us about software.	Ask for trivial changes.	Will not pay for quality but expect it.	Will not pay for quality but expect it.	Understand.	We don't want to know about technology.

Source: [1].

a few to start us off. Some are from colleagues and some are from my own experiences.

- On one project I worked on, testers and developers were not allowed to talk to each other because it would waste time!
- The testers didn't check whether the reports they sent to the developers were duplicates. They did not talk to other testers about what they were reporting. It really wasted the developers' time.
- When I said I had not used a PC before, he treated me like an idiot … they are always so rude in the IT department.
- When I was asked to build the new software, I wanted to find out what I should be building—there was no specification—so I went to talk to the system users to see what they wanted. No one had told them the system was being changed, and they were furious. They spoke to my manager to ask what was happening. Main result: I was reprimanded for talking to people outside the IT department.

Why don't we get along? Maybe it's because we should be talking to each other and listening to each other more.

2.2 Defining the team

To help us understand how to overcome this problem of mutual distrust, let us examine who is in the software team. I will include in "the team" all the stakeholders for quality. This is not only people who commission, design, build, deliver, and support the software, but also people who use or are affected by the software. These definitions are based on my own experience of software projects and customers.

I have divided the team into five groups of stakeholders, based upon their main interests in the software project and, therefore, the definition of quality that they favor. These groups do not map to skill sets or organizational functions, and the people in each group will have various job titles. A person may be in more than one group for a particular project, or may move between groups in different projects. We will see how each group contributes to the success of the software project in a particular way. The groups are:

- People who are customers and users of software: *customers;*
- People who manage software projects: *managers*;
- People who build software: *builders*;
- People who measure software quality: *measurers*;
- People who provide the support and infrastructure for the project and the deployment of software: *supporters*.

In Chapter 1, we saw that there are five definitions of quality:

- Product-based quality is based on a well-defined set of software quality attributes that must be measured in an objective and quantitative way.
- Manufacturing-based quality focuses on the manufacture of software products, that is, their specification, design, and construction.
- User-based quality is fitness for use.
- Value-based quality is focused on things that impact on the running of the business as a whole.
- Transcendent quality can be recognized easily depending on the perceptions and the affective feelings of an individual toward a type of software product.

We will see that each group favors particular views of quality, and this is one of the causes of disunity between the groups.

2.2.1 People who are customers and users of software

Customers include all the people who buy, use, and are affected by software. They include the people who will pay for the software: the project sponsor, budget holder, or purchaser. The people who will use the software belong in this group (see "Users" in Table 2.1). I also include here the customers of the end users—they do not use the software but are affected by it when they interact with the end users.

When I refer to customers later in the book, I mean anyone in this group. They are all stakeholders for the quality of software because they commission, buy, use and, are affected by the software.

This group will hold a transcendent view of quality; as we saw in Chapter 1, this includes one's emotional reaction to a product or service. Their primary quality viewpoint is the user view of quality, so they will want the software to be fit for their purposes. The purchasers will also hold the value-based view of quality, wherein quality is concerned with a product or service being affordable and good value for money. This group is discussed in more detail in Chapter 3.

2.2.2 People who manage software projects

Managers include all the people who control the planning and management of the software project (see IT Managers in Table 2.1). For a particular project there may be a hierarchy of control starting from a project/program board. Reporting to the project board might be a program manager, one or more project managers, and, working under them, project leaders and team leaders. These people are stakeholders for quality because they are responsible for the time and budget control of the project, and for meeting planned delivery requirements. These are important aspects of quality: the customer needs a product they can afford.

This group will hold a transcendent view of quality. They will share with the purchasers the value-based view of quality. They may also support a manufacturing view of quality—an interest in process and identifying defects—especially if they come from an IT development background.

This group is discussed in more detail in Chapter 4.

2.2.3 People who build software

Builders are the people who specify, design, and build the software and other products. This includes the development team (see Developers in Table 2.1). The people in this group include business and system analysts, software architects, designers, software engineers, programmers, developers, technical writers, and trainers. They are stakeholders for quality because they build quality into the product, which includes not just the deliverables (for example, code, user guides, and training material) but also the interim products (for example, requirement definitions, designs, and specifications).

This group will hold a transcendent view of quality. Their primary quality viewpoints are the manufacturing and product-based view of quality. The product-based viewpoint measures quality against the attributes of the product or service; its performance and reliability characteristics, for example. This group is discussed in more detail in Chapter 5.

2.2.4 People who measure software quality

Measurers include people who check the conformance to and suitability of processes, as well as those who check the quality of the products, including software (see Quality Practitioners and Testers in Table 2.1). This group might include the quality assurance (QA) teams (e.g., audit and compliance) and quality control (QC) teams (e.g., test and inspection). By QA I mean processes and activities that check the suitability of and adherence to processes. QC, in contrast, includes processes and activities that check products for completeness, correctness, suitability, and adherence to specification. The QA activities might include process review and quality or process audit. The QC activities include software testing, software inspection, and product review. The membership of this group may be drawn from the other groups or they may be specialists. Beware of believing that this group improves the quality of products; in fact, what they do in this group is measure quality and provide information to the other groups. The measurers support decision making and quality improvement. You may be surprised by this idea, but consider this: the builders build quality into the products, the measurers measure the quality of the product. I will enlarge on this discussion in Chapter 6. This group will hold a transcendent view of quality. They will also favor the manufacturing and product-based views of quality. Some people in this group work mainly in user acceptance testing and will support the user view of quality. The group is discussed in more detail in Chapter 6.

2.2.5 People who provide the support and infrastructure for the project and the deployment of software

Supporters have an important part to play in achieving software quality. They are involved in two areas. First, this group maintains the software when it is delivered and accepted by the customers. They provide support and infrastructure and are therefore stakeholders for quality in that they will have requirements for the software. These relate particularly to its quality attributes: security, performance, portability, and maintainability will all be factors for this group. Second, this group will provide the support and infrastructure for the other groups. They will supply the environments for the building and testing of the software and will support the tool sets used by all the groups in their work. The group includes the IT infrastructure team, comprising IT operations, support, and maintenance; IT security; the help desk; service management; networking; and database administration (see IT Infrastructure Staff and Software Maintainers in Table 2.1).

This group will hold a transcendent view of quality. Their primary focus for quality is the product-based view of quality, and they will also hold a user-based view of quality, especially as related to service levels. The group is discussed in more detail in Chapter 7.

2.3 Interaction between the groups and within each group

2.3.1 Differences in quality viewpoints

Because of the difference in quality viewpoints across the groups, and because different groups bring different expertise, there should be interaction between these five groups before, during, and after a software project. Each person will identify with one of the groups, and, for a particular project, may take roles drawn from one or more of the groups. If we look again at Table 2.1, we see the evidence for mistrust between the groups. I suggest from my own observations two possible reasons for this unfortunate state of affairs.

First, among other differences, the groups hold different viewpoints of quality (see Table 2.2) and so will contribute effort toward achieving their

Table 2.2 Views of Quality across the Groups

	Group				
Quality View	*Customer*	*Manager*	*Builder*	*Measurer*	*Supporter*
Transcendent view	✓	✓	✓	✓	✓
User view	✓			(✓)	✓
Value view	✓	✓			(✓)
Product view			✓	✓	✓
Manufacturing view		✓	✓	✓	

✓ is a primary quality view.
(✓) is a quality view which may be taken by some people in this group.

own view of quality and not value the other groups contributions and opinions.

Second, if all the groups are not explicitly involved in the project from inception to postdelivery, although they are all stakeholders for quality, they do not interact and communicate effectively throughout the project. How does this lead to unproductive conflict? We have already acknowledged that each group wants to do a good job, but as each group pursues its own definition of what a good job entails in isolation, conflict between the groups becomes inevitable. Conflict increases as feedback between groups becomes negative or ceases altogether:

> The main cause of mistrust between groups comes from poor motivational dynamics. People's jobs are not designed to create constructive feedback between groups or members of groups. Individuals usually hear only when things go wrong and rarely get feedback to understand and learn from their successes. The outcomes are groups that may be in conflict, or a group is marginalized in terms of project involvement. Both are highly demotivating and can lead to serious under-performance. (Richard Warden, personal communication, April 24, 2003)

It is possible to address and resolve both these problems. Essentially, it is a matter of communication. To overcome them, we need to provide the means to improve communication between groups, so that each hears and understands the others' point of view. Each group will interact with all the other groups (Figure 2.1), either by direct or indirect communication, or because of views held from previous encounters. Some of these may be positive viewpoints and some may be negative.

Warden and Nicholson [1] particularly noted that groups attribute problems to people outside their own team. These biased attributions of faults are summarized in Table 2.3, based on the MIP Report, and show that "No matter how badly my group behaves or how well the other group behaves, I will see minor faults outside my own group before I see major faults within it. The other side is doing the same thing" [1]. The table summarizes how people react to others; in general we tolerate quite bad behavior from "our

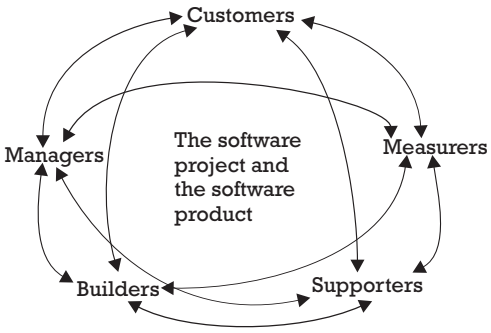


Figure 2.1 Interactions between groups during a software project.

Table 2.3 Attribution of Faults and Problems Inside and Outside "My Group"

If someone in my group behaves well,	I make a positive attribution. I attribute their good behavior to personal goodness.
If someone outside my group behaves well,	I look for a negative attribution. I attribute their good behavior to ulterior motives.
If someone in my group behaves badly,	I struggle to make a positive attribution. I make excuses for their bad behavior.
If someone outside my group behaves badly,	I make a negative attribution. I attribute their bad behavior to personal badness.

gang," but are quicker to take offense at the behavior of other groups. This is human nature and we must struggle to overcome it.

Each of the groups claims that they are motivated to improve performance. However, as each group has its own set of priorities and definition of quality, each often hears the others' views as if they were in direct opposition to their own. For example, Warden and Nicholson [1] found that infrastructure staff measure quality as reliability and service level. They consider the pursuit of the latest technical excellence as an attack on quality, perhaps because use of "leading edge" technology will be more difficult to support. At the same time, developers may see this as a demand for more primitive, hence, less excellent, technology because they measure by product attributes, and perhaps by a transcendent view of excellence. Both sides assume that the others' aspirations are equivalent to their own fears.

Do the quality, process, and excellence models that we looked at in Chapter 1 help us with this problem?

2.3.2 Intergroup relationships in CMM® and Personal and Team Software Processes

In the Personal Software Process [2], the emphasis of the process described is on the individual software engineer (the builder). Individuals are asked to plan and track their tasks and time in order that they can make and meet commitments made to others. This individual self-discipline is an essential ingredient in the success of the software project as a whole. In the Team Software Process [3], the team is made up of software engineers, each of whom is following the PSP, and is therefore using CMM® Level 5 processes; it has been used with "pure software teams and with mixed teams of hardware, software, systems, and test professionals" [4]. The PSP and TSP do not include the customer (see Table 2.4).

The software engineers must interact with their customers in order to understand the requirements, budget, and timescale, but the customer is not seen as part of the team. It is often true as well that the support group (IT infrastructure, operations and so on) are only involved in a minimal way.

This isolation of the software engineers from other groups is very common in software project teams, but is it counterproductive? I would suggest that it is. I have noticed that IT groups and software engineers are not well

Table 2.4 PSP, TSP, and the Groups

PSP	*TSP*	*Matches to:*
Software engineer	Software engineer	Builder, Measurer, Manager
Does not include	Does not include	Customer
Does not include	Infrastructure	Supporter

regarded by many of their customers. Why is this? Partly it is the number and effect of defects in delivered software; this is the problem that software engineering processes such as CMM®, the PSP, and the TSP are designed to overcome.

However, there are significant problems in communication between the groups. On occasion, I have asked software customers their view of IT group. Typical responses are "arrogant," "do not listen," and "in a world of their own." If you ask IT people about their customers, typical responses are "ignorant about IT," "do not listen," and "in a world of their own." Do we see a pattern here?

In my experience, the more the customers and supporters are involved in the project, the greater the mutual understanding of the groups, the better the project, and the more well received the software product. What is important is the interaction between people. At the European Software Testing and Review (EuroSTAR) Conference in 2002, one speaker remarked that although we talk about the HCI (human–computer interface), all the interfaces are human to human—the people who designed and built the software and the people who use it [5].

2.3.3 Intergroup relationships and excellence frameworks—the EFQM Excellence Model

The EFQM Excellence Model (see Chapter 1 and [6]) has four criteria that are relevant. These are people (i.e., employees, the team doing the work), customers, partners (i.e., second or third parties, suppliers), and society. These are matched to the groups I suggest in Table 2.5. A similar grouping is found in other models, for example, the Malcolm Baldrige model [7], which is the equivalent model in the United States.

Table 2.5 EFQM Excellence Model and the Groups

EFQM Criteria	*Matches to:*
People—Enabler	Builders, managers, measurers, and supporters
People—Results	
Partners—Enabler	Supporters (e.g., infrastructure)
	Builders (e.g., third-party software house)
	Measurers (e.g., third-party test services company)
Customers—Results	Customers
Society—Results	Customers—people affected by the software

The EFQM Excellence Model does not tell you how to organize these groups or how to improve communication between them, but it does expect improvement in the perceptions each has of the organization, and of the performance measures of the organization in relation to that group.

It is important for the software team (including all five groups) to consider responsibilities and ownership for tasks and problems. For example, Obeng [8] points out that poorly focused communication between project leaders and project sponsors can lead to a vicious circle of ineffective risk management and increased time spent on getting projects back on track. He suggests actions for the project owners to improve their communication and control. Similar charts of miscommunication and improved communication could be built for other role pairs in the project.

I am not suggesting that everyone has to agree the whole time! For example, the groups' different expertise and viewpoint for quality should be welcomed, as it provides a balance. Some conflict between groups is good!

> In an environment of honest dialogue, the groups constructively challenge each other. In most software situations, the interests of the customers are best served by balancing several definitions of quality. Unchecked, a dominant group will follow their own definition to an extreme. In effective software teams, the supporters ensure that the technologies the builders favor can be maintained. The managers guard the customer's budget by preventing the developers' overengineering and the measurers being inappropriately cautious. In a healthy project, the builders challenge the customers' requirements, pointing out attributes that they haven't considered. (Frank Johnstone, personal communication, April 18, 2003)

What we need are ways to improve our interrelationships and communication. There are a number of techniques and tools that can help with improving communication and fostering understanding between people. Some of these are summarized in Table 2.6, and there is more information in Appendix A.

These methods of improving empathy and communication between people are vital because each of the five groups contributes to and benefits from the software. Each group will require information from the other groups in order to progress and make decisions, but if you are trying to introduce these ideas, you may find resistance. Remember we can only change our own behavior, but how we behave to others will influence how others treat us. In the next five chapters, I will explore each of the groups in turn, suggesting what they can contribute to the project and what they require from other groups during the project. Then, in Chapters 8 to 12, I will look at each stage in the software project life cycle and how each group contributes at each stage. In these chapters, I will refer to techniques for improving and understanding communication, such as those in Table 2.6. I will provide more explanation as particular techniques are demonstrated, either in later chapters or in Appendix A, together with sources for further information.

Table 2.6 Summary of Techniques for Improving Communication Between People

Subject Area	*Technique Examples*	*Brief Description (see Appendix A for more)*
Motivation measurement and job design	MIP [1]	MIP is based on the Job Characteristics Model of Motivation [9]. The job diagnostic survey provides a comprehensive set of motivational measures. As a process model, it can diagnose problems with motivational dynamics caused by poor job design. Psychometric measurement techniques do not provide this capability.
Team relationships and natural roles/ team skills	Belbin team scores [10]	Teams need to understand their strengths and weaknesses as a team. A balance of roles/skills is required in the personalities in the team. Example roles: plants have new ideas, completer–finishers want to finish to fine detail. Too many plants and you will never finish anything.
Improve communication—Empathy with others	MBTI [11]	Different people have different personalities and communication styles. People who wave their arms around and talk a lot can annoy people who like to be quiet and think, and vice versa. Myers-Briggs Type Indicator (MBTI) identifies four contrasting type pairs (e.g., Introvert/Extrovert) leading to 16 "types" (e.g., INTJ is Introvert-iNtuitive-Thinking-Judging).
	Honey & Mumford Learning Styles [12]	Honey & Mumford Learning Styles Questionnaire identifies preferred learning styles (e.g., Pragmatists and Theorists require different experiences to learn).
	Kirton Adaptors and Innovators [13]	Kirton identifies preferred problem-solving methods (Adaptors versus Innovators)—do we break the rules or work within them?
Motivation	Maslow Hierarchy of Needs [14]	Until someone's basic needs (e.g., food, shelter) are met, this is likely to be all they are interested in. Once they have enough at one level, then other motivators become more important. The cutoff point for moving from one point in the hierarchy to another is different for different people.
Improve meetings	De Bono's Six Thinking Hats [15]	Improve meetings by setting rules for behavior. Six "hats" are used. Everyone wears the same color hat at the same time. Example roles: Black Hat—pessimistic, Yellow Hat—optimistic, Red Hat—feelings, White Hat—facts. Allows meeting members to move outside their stereotypes and allows time for different, sometimes difficult, types of communication.
Helping groups agree on goals, aims, objectives, targets, and indicators	Weaver Triangle [16]	On a one-page diagram, the group identifies and agrees on the aim of the project (why it is being done) and associated indicators of success, then the objectives of the project (what is to be done) and associated targets. This helps identify where stakeholders have different aims for the project.
Identify problems and root causes, find solutions	Ishikawa fishbones [17, 18]	Use to identify problems, root causes of problems, and solutions. On a fishbone diagram, brainstorm problems, their possible causes, their root causes, and, therefore, solutions to the root cause.
Walkthrough reviews	Reviews [19]	A type of review with the purpose of increasing understanding of a document. The author introduces the audience to the document and takes them through it, explaining the content.

References

[1] Warden, R., and I. Nicholson, *The MIP Report—Volume 2—1996 Motivational Survey of IT Staff,* 2nd ed., Bredon, England: Software Futures Ltd., 1996.

[2] Humphrey, W., *Introduction to the Personal Software Process,* Reading, MA: SEI, 1997.

[3] Humphrey, W., *Introduction to the Team Software Process,* Reading, MA: SEI, 2000.

[4] Software Engineering Institute Web site http://www.sei.cmu.edu, accessed April 2003.

[5] Hatton, L., "Quantifying Test Value: Some Examples and a Case Study," *EuroSTAR Conference,* Edinburgh, Scotland, 2002.

[6] European Foundation for Quality Management Web site, http://www.efqm.org, accessed August 2003.

[7] Malcolm Baldrige model, http://www.quality.nist.gov/index.html, accessed August 2003.

[8] Obeng, E., "It's Nobody's Baby," *Project Manager Today,* Vol. XV, No. 3, March 2003.

[9] Hackman, J. R., and G. R. Oldham, *The Job Diagnostic Survey: An Instrument for the Diagnosis of Jobs and the Evaluation of Job Redesign Projects,* Technical Report No. 4, New Haven, CT: Yale University, Department of Administrative Sciences, 1974.

[10] Belbin Associates, "Belbin Team Roles," http://www.belbin.com/belbin-team-roles.htm, accessed October 2003; also for assessment information.

[11] Team Technology, "Articles," http://www.teamtechnology.co.uk/articles.html, November 2003.

[12] Honey, P., "What Are 'Learning Styles'?" http://www.peterhoney.com/product/learningstyles, accessed November 2003. PeterHoney.com, 10 Linden Avenue, Maidenhead, Berks, SL6 6HB. Tel.: 01628633946. Fax: 01628633262. E-mail: info@peterhoney.com.

[13] McHale, J., "Innovators Rule OK—Or Do They?" *Training & Development,* October 1986, http://www.kaicentre.com/.

[14] Gywnne, R., "Maslow's Hierarchy of Needs," http://web.utk.edu/~gwynne/maslow.HTM, November 2003.

[15] de Bono, E., *Six Thinking Hats®*, New York: Penguin, 1999.

[16] Evans, I., "The Troubled Project—Best Practice from Theory to Reality," *EuroSTAR Conference,* Stockholm, Sweden, 2001.

[17] Robson, M., *Problem Solving in Groups,* Hampshire, England: Gower, 1995.

[18] TQMI, *Problem Solving—Tools and Techniques*, Cheshire, England: TQMI, 2001.

[19] IEEE 1028™ Standard for Software Reviews, 1997.

Selected bibliography

Honey, P., and A. Mumford, *The Learning Styles Helper's Guide,* Maidenhead, England: Peter Honey Publications, 2002. PeterHoney.com, 10 Linden Avenue, Maidenhead,

Berks, SL6 6HB. Tel.: 01628633946. Fax: 01628633262. E-mail: info@peterhoney.com.

Kroeger, O., J. M. Thuesen, and H. Rutledge, *Type Talk at Work: How the 16 Personality Types Determine Your Success on the Job,* New York: Bantam Doubleday Dell, 2002.

Maslow, A., *Motivation and Personality*, New York: Harper and Row, 1970.

Mullins, L. J., *Management and Organisational Behaviour*, 5th ed., New York: Financial Times/Pitman Publishing, 1999.

Rothman, J., "Team Building at Work," *STQE,* July/August 2003, p. 64.

CHAPTER

3

Roles and Quality: Customers

Contents

In this chapter I shall:

- Introduce the members of the customers group and their roles and activities;
- Introduce their quality viewpoint;
- Provide a framework for customers' activities within the EFQM Excellence Model;
- Identify information flows between customers and the other groups.

"Why do so many IT people think the world was started with a requirements catalog? Don't they understand there's more to my business than that. . . . I want them to understand what makes my business tick. . . ." "And why are they so arrogant and rude? Personally I'd take on the guy who has fewer technical skills if I thought he understood my world, and he treated me like a human being."

—Two customers for IT services compare notes on just why they hate their IT departments

3.1 Introducing the customers

If it were not for customers, we would not build IT systems. Software is only there to solve people's problems and help them carry out tasks more easily. The context for the tasks might be in home computer use, for example, to play a game or shop on-line. It might be a work context, for example, to reorder stock or produce a report. The use of the computer may be essential to the task, for example, to carry out complex calculations or to control a remote robotic arm in a dangerous environment. It may be intended to make the task easier, for example, intraoffice communication.

The customers for IT systems do not love their IT suppliers or the software. Software is only a tool, and if it is technically excellent but not seen as useful, it will be rejected. In one organization I visited, a staff satisfaction survey showed that almost all the problems associated with staff dissatisfaction were perceived to be caused by poor IT provision. We saw in the quote above that customers have various reasons for dissatisfaction; the IT work itself may be poor, and the IT people may not be liked, either. IT people are seen as self-satisfied and arrogant; customers may feel sneered at if they do not understand the technical jargon. The second customer in the quote above told me that the IT support person they chose eventually for their team was the least well qualified technically, but was the politest and the one who was able to work as part of the team. In order that the customers receive the software they need, they have to communicate with those who supply the software, either directly or by representation; communication is vital to success.

Not only do the customers observe that IT "solutions" fail to solve their problems, they often believe that software in fact *adds* to them, for example, by adding steps or by hiding information. Additionally, if IT people are focused on providing functionality against a requirements list, they may lose sight of *why* the customer wanted the software.

When I visit large IT organizations, it is easy to see who in the IT department is well regarded by the customers; look for the queues by particular desks and you will know that those are people who have troubled themselves not only to understand IT technicalities but also to understand their customers.

In this chapter, I will examine who the customers of IT systems are, what they want, and how the communication gap between IT people and their customers can be bridged.

3.2 Who could be in this group?

For customers, the software is a tool, a means for them to perform tasks efficiently and effectively. This group includes:

- People who will pay for the software, for example, the project sponsor, budget holder, or purchaser.
- People who will use the software delivered (see "users" in Chapter 2, Table 2.1), for example, the customer service agents working at the call center of a financial services organization also request a change to the software they are using to match a new business process.
- *The customers of the end users:* They do not use the software but are affected by it when they interact with the end users; for example, someone who contacts the call center to make a change to an investment may comment on the services provided, and this may lead the customer service agents to request a business process change, which leads to a software change.

- *Society and government as customers:* There may be a legislative change that requires change in business processes, supporting systems, or both.
- *Organizational customers:* Other people in the organization who need particular outcomes; for example, the finance group may want to reduce costs and so request streamlining changes.
- In IT organizations without a direct customer, for example, those companies building and selling packages for commercial or home use, the marketing or product design groups may commission software and are customers in that sense, but they will not use the software.
- The IT organization itself may request changes; for example, the operational support group may wish to improve the maintainability, performance, or reliability of IT systems without changing the functionality and usability for the customer service agents.

These people are all stakeholders for the quality of software because they commission, buy, use, and are affected by the software.[1]

Let us examine some different types of customers and their relationship with software and with other groups.

3.2.1 In-house customer

In this situation, the IT department works for the same company as the people who will use the software and those who will pay for it. The managers, builders, measurers, and supporters may have a captive audience of customers who must use their services, or they may need to bid for projects if the customers choose to put work out to tender. I have seen this happen, with the IT department bidding against external third parties, and with teams within the IT department bidding against each other for projects.

When the customer is in-house, the relationship between the groups can become very close. The managers, builders, measurers, and supporters have the chance to understand the customer and the organization in enormous detail. At best, this has the advantage that the IT people can provide a real depth of service to the customers, who, in turn, can become involved easily throughout the process of commissioning, building, delivering, and using the software. Careful choice of processes and methods, as described in Chapters 9 to 12, can allow the customers' constantly changing business needs to be met because all the groups are in the same organization and should, therefore, have the same overall strategic goal. One IT customer remarked to me:

> If staff turnover in customer departments is much higher than in IT, the builders and supporters may have *more* business knowledge, which enables them to help the customers identify how software can create profitable

1. The IT Infrastructure Library (ITIL) [1] differentiates customers who pay for the software, and users who actually use the software. In this book, I have included both those groups together as customers.

opportunities. This can, however, become detrimental to the relationship if it reaches the point where the IT people are dismissive of the customers and feel it is their role to *tell them* what they want.

So we see that it can be disadvantageous if the IT people become complacent about their customers, leading to the types of comments we saw at the start of the chapter. The costs of work can be hidden; the IT work may be seen as "free" because no invoices change hands. Alternatively, if work is charged by a system of internal billing, with customers having a budget to use for IT work and the IT group having charging rates to bill against the budget, then both the customer and the other groups can monitor the cost of the software, including the cost of changes. Communication difficulties can arise because of the personality and cultural differences between the groups. I am struck by how often customers complain that the others, especially IT specialists, whether managers, builders, measurers, or supporters, are uncommunicative, arrogant and rude. Meanwhile, IT specialists complain that customers do not know what they want, and also misunderstand the communication style of their customers. In one organization, I remember the business users remarking on the difficulty of getting what they needed from the network support specialists; the technical team would not respond to questions, and often did not even reply if spoken to directly. One day, I asked a member of the technical team if he thought there were communication difficulties across the teams. "No," he said, "the network has been up 100% this week." So, I changed my question to "Do you talk with the business people very often? Do they come and ask you when they have a technical problem, for example?" He said that they did come and ask if they had problems and that he would deal with the problems, but that the business users tended to chat, fuss, and waste his time. What I observed taking place were dialogues like the one in Table 3.1.

In truth, the people on the technical team didn't see the need to provide feedback at the level their customers required it. How do we deal with this problem? The Myers-Briggs Type Indicator (MBTI) [2] seems to show that different personality types require and give different types and amounts of feedback during communication. This means that we need to try and

Table 3.1 Typical Dialogue Between a Technical Team and Its Customers in One Organization

Tom Customer says:	"Hi—how are you? I'm trying to do my e-mail and I can't get access to the network—it just doesn't seem to be working and . . . "
Ted Technical thinks:	*I'd better look into that.*
Tom Customer thinks:	*Did he hear me?*
Ted Technical	Solves problem but does not tell anyone
Tom Customer, some time later:	"Is the network available yet?"
Ted Technical replies (irritated):	"Yes, of course it is."
Ted Technical thinks:	*Why does this person ask me stupid questions?*
Tom Customer thinks:	*Why is this person so rude and unhelpful?*

understand other people's communication styles. As customers, we need to consider whether we have communicated in a way that puts our message across, and that results in the feedback we require. The dialogue could be managed differently (see Table 3.2). This improved dialogue allows Tom Customer to know what is going on, while acknowledging Ted Technical's desire to minimize "unnecessary" conversation.

3.2.2 Third-party custom-made system customer

Here the customer asks another company to provide a software system, and the supplying company is going to design and build the software to meet the customer's specific requirements. Like a custom-made suit, it will be made to fit the particular customer rather than being bought ready-made. This allows the customer to ask for tenders from a number of suppliers, and to make a choice based on reputation, confidence, ease of relationship, understanding of the customer's organization and needs, expertise in a particular application or software type, standards followed, price, or whatever the key factor is for the customer organization. This allows the customer to pick the most suitable supplier to meet their needs and to pay them on a partnership or contract-by-contract basis.

The third party will be keen to get repeat business and this should focus them on making every effort to meet the customer's changing needs; they will not be complacent about their position as a supplier. However, if the customer and supplier are working to a fixed-price contract, the supplier will resist change if this means that the service or product will be delivered at a loss. We will see in Chapter 9 how much work is required to reach the point where we can agree on a contract for a software development life cycle (SDLC), and the importance of involving all parties in understanding what is required. For a third-party supplier, the risks associated with agreeing to a fixed-price contract when there are uncertainties are enormous, so customers need to be consider what type of contract is best not only for them but also for their suppliers. As one customer manager remarked to me:

> A fixed-price contract that is not competitive to the supplier does not necessarily make good business sense for the customer. Third parties have quit punitive fixed price contracts, forcing the customer to reissue for tender. On occasion, the delay caused has prevented the customer from realizing the benefits predicted from the software.

Table 3.2 Improved Dialogue Between a Technical Team Member and a Customer

Tom Customer says:	"I can't get access to the network. Will you look at it please?"
Ted Technical says:	"Sure, give me five minutes to finish this reboot."
Tom Customer says:	"No problem. Give me a shout when the network is running, please."
Ted Technical says:	"Yup!"
Ted Technical	Solves problem and shouts across, "OK now!"
Tom Customer says:	"Thank you!"

Options for a single project might include a time and materials contract; time and materials to a budget ceiling; a series of short investigative projects to reach a decision about the content of and type of contract; or agreement to use an evolutionary, iterative, or incremental approach (see Chapter 10 for an explanation) in which the budget and scope could be renegotiated on a stage-by-stage basis.

Some customers prefer to build a partnership relationship with their third-party suppliers, rather than working on a project-by-project basis. This can be more satisfying for the customer and for the supplier, as relationships build up over time; the supplier's staff feel like real stakeholders in the customer's success, and they can see the outcome of their work, which increases their satisfaction and motivation. In turn, this means that they are happy to work with the customer, and, therefore, provide a better service to the customer, who in turn reports increased satisfaction with the supplier.

A closely embedded relationship with a third-party supplier can lead to similar advantages and disadvantages as the relationship with in-house IT groups that we discussed in Section 3.2.1, but, additionally, if the customer's circumstances change very significantly it can be difficult to disentangle the two organizations. Communication difficulties can arise because of cultural differences between the organizations, as well as because of personality differences between the groups and individuals. Often, a supplier organization will put its most "customer-friendly" people in the jobs that require customer contact. This improves the customer–supplier interactions, but may have the effect of moving the dialogues in Tables 3.1 and 3.2 from the customer organization to the supplier organization.

With third parties, one great advantage is the "fresh look"; sometimes, an outsider can see our strengths and weaknesses and how to improve more clearly than we can ourselves.

3.2.3 Third-party package or commercial off-the-shelf (COTS) customer

Sometimes, instead of ordering a custom-made solution tailored to our specific problem, we may choose a "ready-made" or "off-the-shelf" product. This is referred to as commercial off-the-shelf (COTS) software. The customer should be able to buy, install, and run this software without any tailoring or changes built especially for them, although there may be options they can set within the package. In this case, the relationship between the customer and supplier may be quite limited or consultancy and support may be offered as part of the purchase. It is likely that the builders and measurers never meet the customer directly. Instead, the customer relationship is managed via the sales, marketing, support-line, or client-management functions within the supplier. For the customer, the contact is minimal, perhaps limited only to the support line when problems arise. However, customers can influence what is built into future versions of the package, for example, by taking part in user groups, lobbying suppliers for change, and reporting problems or improvement suggestions.

Many different types of customers will become involved with COTS software, such as the following:

- *Large organizations:* Such customers are likely to have their own IT departments or commission custom-made software; however, they will probably choose COTS to automate those generic business processes that do not give them a competitive advantage and so do not need to be different from their competitors' processes. We see this in the success of some payroll and accounting packages, which are used across many industries. In such COTS acquisitions, the large organizations normally change their business processes to fit those required by the package, this being seen as a more economical solution than commissioning custom-made software.
- *Small businesses:* Many small businesses do not have IT departments and cannot afford to engage third parties to build custom-made software; they are likely to buy COTS. Companies can get competitive advantage from COTS by using it differently to their competitors. For example, one colleague reported to me that several years ago that one small importer became the first participant in the (then) niche active sports goods market to use a stock control system that was standard in more mature sectors. For a while, this gave the company an edge over its rivals.
- *Home and hobby users:* These customers use software for entertainment, socializing, pursuing a hobby, or doing chores. Sometimes they buy COTS to change the way they do an existing activity, for example, using e-mail to write to friends or using a spreadsheet to manage their personal finances. Most computer games are, of course, creating a *new* need.
- *The niche user:* There are very specialized COTS systems [3], for example, those used in military applications. You may see these referred to as NOTS (niche off-the-shelf) or MOTS (military off-the-shelf).[2]

With COTS suppliers, one of the supplier's problems is that sometimes there is no clear customer group; sales, marketing, client management, and the support line or help desk act as a proxy customer for the other groups. The customers' goals and requirements are found by market research, or by looking at complaints and improvement suggestions. Market researchers interview groups of likely or existing customers to help focus the supplier's understanding of the customer's viewpoints. However, the customers will see these immediate contacts—the sales, marketing, client-management, or help-desk teams—as the face of the supplier, and direct any frustration with the quality of the software at them. As a result, these contacts may become builders by proxy!

2. Definition copyrighted and used with permission of whatis.com (http://www.whatis.com) and TechTarget, Inc.

We mentioned earlier that games programs are examples of an entirely new product; before a computer game is invented, people do not know they need it. What happens when a product is entirely new? We may not, as customers, know that we need it; we have not specified requirements, and we have not noticed a need, we do not know we have a problem. I have seen cases in which one of the builders, a developer, or software engineer has invented a piece of software and then asked, "What could we use this for?" or has identified a way to solve a particular problem. Development of the ideas will then be led by sales and marketing based on their knowledge of the end customers' likely buying habits, and by market research.

3.2.4 The IT specialist as customer

Interestingly, people from all the groups become customers at some point, because all the groups use software tools to aid them in their work. For example:

- Managers use planning and reporting tools.
- Builders use compilers and software engineering and support tools.
- Measurers use test and review tools.
- Supporters use operational support tools and networking monitoring tools.

It is always instructive to see members of the IT group in receipt of a software tool, complaining about the lack of help messages, user guide, usability, functionality, and so on, and then to look at the customer reports on their last release. The IT specialists may also request changes to existing software in order to make it easier to support, for example, changes to make it more reliable or have better throughput. User and business customers must be involved in these projects from the start-up discussions, even if they are not the originators of the request for change. This is because any change may have an impact on them; for example, I remember a performance upgrade for a system that theoretically did not change the functionality but, in practice, resulted in a complete retest of functionality because the code had been entirely rewritten. This was decided on by the builders of the system, but the user and support-line groups had not realized the amount of testing that would be required from them, as they had not been involved in the discussions at the start. As a result, the project was late in completing because the system and acceptance tests took longer than expected and resulted in more problems than expected.

3.3 Quality viewpoint

The customer perspective is about supporting the needs of the individual customer, but also those of the organization and its stakeholders. These

needs must be met within the individual or organization's constraints, whether of time, money, or expertise.

The user-based definition says that quality is fitness for use. Software quality should be determined by the user(s) of a product in a specific situation, either in a home or a business. Different customer characteristics require different "qualities" of a software product. This can be subjective and cannot be determined on the basis of quantitative metrics alone. It is the user-based definition that encourages us to validate *as well as* verify the system. For example, fit for purpose might mean that I can do my work efficiently and effectively when I use this software.

This group may hold a transcendent view of quality. As we saw in Chapter 1, transcendent quality is based on an emotional response to the product. For the supplier, especially of COTS software, the transcendent qualities will be very important. What is it that makes one software package more appealing than another? Different customers will hold views based on fashion, culture, prejudice, and previous experience. Customers need the answers to these types of questions:

- When will I get it?
- Will it do what I want?
- Can I afford it?
- Can I rely on it?
- Will I enjoy using it?
- Will it make my life easier?
- Will it support our improvement strategy?

3.4 Quality framework using the EFQM Excellence Model

3.4.1 The EFQM Excellence Model and the customer organization

For a business customer, software is used to help in tasks that allow the organization's goals to be met. In the EFQM Excellence Model (see the description in [4], for example) information systems come under the heading of "partnerships and resources," that is, information and information systems are resources that enable the organization to achieve its strategic goals.

Similarly, if we look at an organizational measurement system, such as the Balanced Scorecard, information systems are seen as critically important tools that enable employees to work effectively, so that they can contribute to the goals of the organization. We will look at how managers use the Balanced Scorecard in Chapter 4.

This means that the customer's quality framework should influence the quality framework used by the other groups. For example, in one software company that provides third-party, custom-made software to its customers, the software company's quality manual says that a project's standards

should always at least equal the customer's. If the customer has processes or practices that exceed the software company's standards, those will be adopted for a project, but if the customer's processes are less rigorous than those of the software company, then the software company's processes will be used.

The other groups need to be aware of the customer's strategy and quality framework, including any measurements and targets, in order that they can deliver services and products that help the customer meet the organization's quality targets. The emphasis will change, and this will change what constitutes quality for the organization. The organization may be driven by a need to meet a particular external standard, to keep within legislative or regulatory bounds, to increase market share, to reduce time to market, or to be a world leader for excellence of service. The supplier's ability to recognize and adapt to the customer's quality framework is a critical success factor in the effectiveness of their relationship. The framework for the customer translates into departmental goals and, finally, into the personal objectives and targets for an individual system user, but it also translates into a complementary subset of the supplier's goals and, hence, into personal objectives and targets for managers, builders, measurers, and supporters. This quality framework will drive what the customer needs from the software.

In Chapter 1, we looked at the EFQM Excellence Model and how it is divided into nine parts: five enabling criteria and four criteria for measuring results. In Section 3.4.2, we will look at how the EFQM Excellence Model enablers could be interpreted for the customers of an IT project, and in Section 3.4.3, we will look at customer results. Remember that the EFQM Excellence Model is based on the fundamental concepts of excellence we discussed in Chapter 1, and that equivalent models such as the Baldrige model are available.

3.4.2 EFQM Excellence Model enablers for customers

3.4.2.1 Leadership

Leadership for customers is the leadership of their organization. If the vision and goals of the organization have not been communicated to the specific customers who are commissioning software, then they will not commission software that meets the quality goals of the organization. The pressures on organizations will change the goals for the customers. Lately, for example, issues like information security and IT governance are playing an increasingly important role in organizations. Now, leadership in IT governance is expected from the board of directors and the executive management, so it is integral to enterprise governance. Leadership from the top is required to ensure that the organization's information systems sustain and extend the organization's strategies and objectives. Leadership from the top is also required to ensure that information and systems are secure [5].

3.4.2.2 Policy and Strategy

The policies and strategies for the organization reflect the vision of the leaders. Customers of IT software and systems need to consider the policy and strategy for acquiring software that supports their information, improvement and organizational goals. As customers, we implement IT strategies to reflect organizational strategies; for example, different customers might need IT strategies to:

- Support the work in a particular industry sector, for example, different strategies for supplying the avionics industry, education sector, or home computing.
- Support a marketing need, for example, different strategies to support quick time to market with new products, high market demand, increased market share, or emphasis on a small number of high-value customers.
- Support the administration of the rest of the business, for example, accounts and payroll.

Policies are brief statements that are the basic rules for how an organization conducts itself. As customers for IT, we want policies for obtaining IT systems that enable us to fulfill our strategies. For example, different customers might have policies about:

- Acquisition types—"We will only buy COTS."
- Supplier choice—"All suppliers must have achieved at least CMM® [6] Level 3."
- Supplier quality processes—"All acquired software must be designed and built complying to standards within the BSI Software and Systems Quality Framework (SSQF) [7]."
- Relating to the supplier and involvement—"We want the supplier to use evolutionary delivery methods and to be involved at every stage."

3.4.2.3 People

The people we are considering here are the customers themselves, because of their part in the software team. As we will see in Chapters 8 to 12, it is useful if customers are involved throughout the life of software, not just as the users of software, but also during the definition, build, and testing of the software. One of the advantages of the evolutionary delivery method, for example, is that the feedback cycle between the customer and the other groups is very fast. The customer is sent frequent improvements to their software, and is also able to feed back to the supplier improvement ideas needed to cope with changing requirements. According to Tom Gilb (at a seminar on Evolutionary Delivery, London, September 20, 2003), this feedback loop enhances the customer's satisfaction with the software delivery.

Customers can improve their contribution to the software quality and their understanding of the software project by learning how to take part in certain key activities. Some years ago, I worked at a software company where I helped to introduce a number of quality processes. We had put together an in-house document review process, based on experiences with walkthrough and inspection methods that several of us had used in different organizations. This process allowed us to make decisions about the type of review based on the risks associated with the document, for example, the value of the contract, the difficulty of the technical work, and so on. The reviews had the general structure of:

- Up to 2 hours preparation before the review to identify problems and suggestions for improvement;
- Up to 2 hours meeting to document those problems and identify new ones, with no more than six people attending, and optionally the second hour being used as a "solutions" meeting.

Time spent by each participant and the number of problems found were logged, so we could check that people were using their time effectively and efficiently. You will see that as a process, this does not match the rigor or structure of, for example, the Gilb and Graham inspection [8], but it was a vast improvement over the previous situation, which was quite unstructured. It was a good enough process. As part of an improvement program, we then decided to invite some of the customer organizations to take part in document review training courses. The benefit to us as supplier and to them as customers was that when we asked customers to review contracts or requirements, we all knew how to do it, how it was going to be organized, how much it would cost, and the purpose of the meeting. The customers enjoyed taking part and gained considerable satisfaction from contributing constructive criticism and seeing it acted on. The software development life cycle and the deliveries went far more smoothly; one of the directors commented to me that he spent far less time firefighting than ever before.

This principle of training customers to contribute more effectively can be applied to a number of activities:

- Training in document reviews. For basic understanding of a review process, but also for review of particular artifacts. For example, "If you are reviewing a use case, expect to see…."
- Training in user-acceptance testing, including an understanding of why user-acceptance testing is different from other types of testing, and what techniques might be applied. For example, "If you are testing a Web site you will need to consider usability and security as well as functionality."
- Training in process audits and reviews, including understanding why, during acceptance, the customer might want evidence of quality activities and how this might be found. For example, "If you are reviewing a supplier's work, ask to have some tests run and review the output from tests."

3.4.2.4 Partnerships and Resources

The customer's partners and resources for the software include all the groups: those who are supplying the software and those who will support it during use. The customers require a certain level of service from the software systems, and this will form the basis for the service level agreements (SLAs) with the supporters group. The criteria that the customers will use to decide whether to accept the software must reflect these SLAs because, in order that the supporters group can fulfill the SLA, the software must be capable of delivering according to the SLA. Therefore, the customers must work with the supporters to set acceptance criteria. We will see in Chapter 9 how to do this.

As well as partnerships with the supporters, customers will have partnerships with non-IT suppliers and their own customers. The software systems may need to share information, or they may need to be protected. One of the customer's acceptance criteria for system attributes may consider how well the delivered software supports control of interfaces with systems outside the customer's organization.

3.4.2.5 Processes

The customer's business processes must be supported by the software and systems. If the systems do not support and improve the organization, there is no point having them. The business processes may be industry-specific or general administration processes. They may also need to meet requirements in particular quality standards, for example, ISO 9000 [9]. The business processes should be designed to enable the customer to meet the organization's strategy and goals efficiently and effectively. Any software systems are there to make the work more effective and more efficient. The other groups need to understand the customer's processes and standards in order to support them.

3.4.3 EFQM Excellence Model results for the customers

3.4.3.1 Customer Results

The customers of the IT systems and software have their own customers. How does the software affect these customers? The customers need to find out how the services they offer to their customers could be improved, and this may include improvements to the IT systems and software. Results of satisfaction surveys, complaints, letters of praise, and comments all indicate where processes and, hence, systems could be improved.

3.4.3.2 People Results

Are the people in the customer group who work with the IT groups happy to do so? Do the others in the customer group see them as representative? It is important that the customer organization understands whether the IT

suppliers are good to work with. Look again at the quotation at the start of this chapter. These customers are not unhappy with the technical excellence of the work done by their IT teams, nor are they questioning the number of coding defects. The two problems identified are lack of understanding of the business in its essence and poor interpersonal skills. In one organization that I visited, the staff satisfaction survey revealed that over 90% of the staff's complaints had to do with poor IT provision. This type of information allows the customers to show the other groups what changes are needed. Many authors, for example, Gilb [10] and Watkins [11], have observed that it is the nonfunctional rather than the functional attributes that cause complaints or cause people to reject a product. This fits in with my practical experiences. Functionality is taken for granted, and is usually delivered, but the real problems come in understanding what level of attributes such as reliability, security, and performance are required or deliverable. For this reason, in Chapter 9 we will focus on nonfunctional acceptance criteria.

3.4.3.3 Society Results

We have already noted in Section 3.4.2.1 that as part of the governance of the organization, the governance of IT systems and their security is increasingly important, for governments and for society as a whole. Customers need to consider what effect their organization has on society, and the view that society has of the organization. Consider how IT systems affect the impact of an organization on society, both the perception that society has of the organization and measures of how the IT systems help the organization to perform.

There will be legislative and regulatory controls set by government that must be obeyed. These include disability discrimination and data protection legislation. When looking at IT systems and projects, customers should consider how these support the organization in making products and services accessible, while protecting privacy. For example, is it reasonable to use a copy of live databases to test a new version of the software? Leaving aside all the technical reasons why this might be a bad idea, from a data protection and security viewpoint is it right to allow detailed access to the data to people who would not normally see it?

It is also useful to consider the effect on society when reviewing risks. In a financial services project, I looked at assessing the impact of problems with a group. We scored impact from 1 to 5, with 5 as highest impact. We set a "typical story" for each score to help us score the risks we were assessing, rather as in Table 3.3.

3.4.3.4 Key Performance Results

Key performance results measure financial results such as profit, return on investment, and turnover, but also measure nonfinancial results such as market share. The customer will want to know the efficiency and effectiveness of the IT systems in their contribution to the overall key performance

Table 3.3 Assessing Impact of Risks—Effect on Society Scores High

Impact Score	*Description*	*Story*
5	External publicity—external customers affected	Chief operating officer appears on a consumer affairs program to explain why the systems have gone wrong.
4	External awareness—external customers affected	Letters and calls of complaint, dealt with individually.
3	Internal publicity—internal customers affected	There are system failures; the call-center staff can work around these, but they are disruptive.
2	Internal awareness—internal customers affected	There are some minor system failures which the call-center staff can work round easily
1	No publicity—no customers affected	IT department affected by failures but can prevent the call-center staff from being affected.

measures. The bottom line question is "Was it worth investing in the software—did we get a return on investment?" We will look in Chapter 4 at how the manager can use measures in the project that tie into the customer's key performance measures; the customer needs to ensure that the manager knows what these are.

3.5 Communication between the customers and other groups

We will see in Chapters 8 to 12 that customers are involved in the whole life span of a piece of software, from its conception, through the software development life cycle (SDLC), during delivery, and postdelivery until decommissioning. We will see that some of the SDLC models encourage customer involvement and other models do not. The advantage of having little involvement to us as customers is that we order the software, and while it is being built and delivered we get on with other things. Superficially, it looks as though we are being efficient with our time. The disadvantage is that during the period of the SDLC, things will change; our problems as customers will change and the solutions we require will change. This means that by the time we get the software, it is out of date. The disadvantage of SDLCs with high customer involvement is that it is time-consuming for the customer, who also has "business as usual" to deal with as well as the SDLC. The advantage is that it is far more likely that a useful software solution will be delivered.

So we will see in the later chapters that communication between the customers and the other groups is needed throughout the whole life span and that each of the groups must communicate with all the other groups. The customer needs to exchange information with all the other groups (Figure 3.1).

In order to decide whether it is worth acquiring new software, for example, the customer needs to listen to the manager's view on cost effectiveness, the builder's knowledge of technical constraints, the measurer's

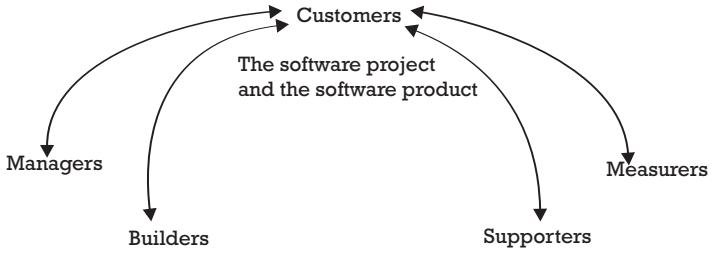


Figure 3.1 Communication between the groups.

experiences of likelihood of problems, and the supporter's view of impact on the existing systems. A risk assessment would be done at this point, and, in particular, customers need to assess the impact on the organization or business if the change is not made, if it goes wrong, or if it is late or over budget. In parallel with the risk assessment, therefore, the customers start to set constraints for cost and time. While doing this, they should be listening to the builders, supporters, and measurers' views about the technical risks and the technical constraints on what is being proposed. An SDLC may be appropriate, but it may not be the best solution. Whatever is decided on, a contract for the work needs to be agreed on. We will see in Chapter 9 how to reach agreement on aims, objectives, targets, and indicators, as well as measurable acceptance criteria. There may be constraints on the accuracy and precision of estimates, and customers should discuss this with the other groups, and expect that estimates may have a broad range. At this stage, particularly in the planning for an SDLC, there will be many things that are unknown, so we would expect to see accurate ranges in the estimates, rather than precision:

- "I can complete this in the next 5 to 50 days" is likely to be an accurate estimate, but it is not a precise one.
- "I can complete this on October 23 at 2:57 p.m." is a very precise estimate, but it may not be accurate.

It is also possible that the estimate makes us realize that it is not possible to build and test what the customer wants within time and budget constraints; something simpler must be agreed on. The customer must encourage honest estimation. Too often, I have seen the IT team attempting to produce an estimate that is acceptable rather than one that is realistic. If experience of the customer's reactions in the past has taught them that they will get praised for a low estimate and criticized for a high one, it is human nature to produce an "acceptable" estimate now, and try to avoid the consequences later. Customers should particularly watch for the situation in which the people making the estimate are not going to be involved in the work; their job is to get the contract, and they may not be focused on the effect of their estimate on the project team. Also beware of fixing the overall estimate too early; once changes are made to the requirements, whether these arise from change in the real world, correction of budget, or correction

of mistakes, this may mean changes to the estimates. Customers will need to negotiate to deal with changes to requirements, whether these have been "set" and therefore require a contract renegotiation, or have been planned for. We know there will be change; we just cannot always anticipate what that change will be. Table 3.4 lists the information that customers have that the other groups need.

Customers need information from other groups in order to understand the constraints on delivering their wish list. Table 3.5 lists the information that customers need from the other groups.

3.6 Summary of the group

Without customers, there would be no software; their involvement and views are critical to the other groups' success. Customers have a vital contribution to make throughout the life of a software system. Their quality viewpoint, of the system being fit for its purpose, and their world, with its changing risks and demands, must influence how the other groups work. managers, builders, measurers, and supporters only exist to enable the customers to achieve the organization's goals.

Table 3.4 Information That Customers Have That Others Need

Before the SDLC starts and updated throughout the SDLC	Which problems/ideas are important to the customer/organization and why
	Changes in priority, new ideas, and problems
	Whether any of the proposed solutions/prototypes are suitable
	Reasons why they are/are not suitable
	What the customer's own customers need from them
	Business constraints (time, cost, process, legal) and why these are constraints risks (impacts) to the organization if we fail or do not deliver the solution
	Whether there are existing workarounds for this problem
	Why this is important; the difference it will make
	How we will measure success postdelivery
	How we will accept the software (acceptance criteria)
	What "fit for purpose" means in this case
During the SDLC	An initial set of detailed requirements
	User-acceptance testing—knowledge, business, budget, and time constraints
	Changes, corrections, and refinements to requirements and acceptance criteria
	Improvement suggestions for software, training, and documentation
	Design of any new business processes
	Delivery plan constraints and postdelivery SLA constraints
	Revisions to the plans
	Agreement on readiness to deliver
	Authorization that the acceptance test has passed or failed
At delivery	Confirmation that delivery is complete and accepted
	Ready to start "real work" signal
Postdelivery	Evaluation of the software in use; is it fit for its purpose?
	Evaluation of the processes used; how could they be improved?

Table 3.5 Information That Customers Need from Others

Managers	Possible solutions to problems/ideas
	Cost and resource constraints and why these are constraints
	Value quality viewpoint
	Whether proposed aims/indicators are understood
	Proposed objectives/targets for the solution
	Whether proposed acceptance criteria are SMART (specific, measurable, achievable, realistic, and time-bound)
	Nontechnical risks (likelihood of this going wrong) and nontechnical constraints such as resources, time, budget, and availability of people
	Precision and accuracy of estimates, when refined estimates will be possible
Builders	Possible solutions to problems/ideas
	Technical constraints and why these are constraints
	Why this problem/idea (for example, for a technical problem) is important and its impact on the customers
	Manufacturing/product quality viewpoint
	Whether proposed aims/indicators are understood
	Proposed objectives/targets for the solution
	Whether proposed acceptance criteria are SMART (specific, measurable, achievable, realistic, and time-bound)
	Technical risks (likelihood of this going wrong) and technical constraints
	Precision and accuracy of estimates, when refined estimates will be possible
Supporters	Possible solutions to problems/ideas
	Technical constraints and why these are constraints
	Why this problem/idea (for example, for a technical problem) is important and the impact on the customers
	Manufacturing/product quality viewpoint
	Shared user quality viewpoint
	Whether proposed aims/indicators are understood
	Proposed objectives/targets for the solution
	Whether proposed acceptance criteria are SMART (specific, measurable, achievable, realistic, and time-bound)
	Technical risks (likelihood of this going wrong) and technical constraints
	Precision and accuracy of estimates, when refined estimates will be possible
	Constraints on SLAs postdelivery
	Scope of supporters' operational acceptance testing
Measurers	Possible solutions to problems/ideas
	Technical constraints and why these are constraints
	Why this problem/idea (for example, for a technical problem) is important and the impact on the customers
	Manufacturing/product quality viewpoint
	Whether proposed aims/indicators are understood
	Proposed objectives/targets for the solution
	Whether proposed acceptance criteria are SMART (specific, measurable, achievable, realistic, and time-bound)
	Technical risks (likelihood of this going wrong) and technical constraints
	Precision and accuracy of estimates, when refined estimates will be possible
	Advice on acceptance testing
	Advice on review processes
	Assurance that QA and QC activities have taken place
	Results of those activities

References

[1] IT Infrastructure Library, *Best Practice for Service Support,* Norwich, England: Office of Government Commerce, 2002, p. 7.

[2] Team Technology, "Working Out Your Myers Briggs Type," http://www.teamtechnology.co.uk/tt/t-articl/mb-simpl.htm, October 2003.

[3] searchCRM.com (via http://www.whatis.com), "COTS, MOTS, GOTS, and NOTS," http://searchcrm.techtarget.com/sDefinition/0,,sid11_gci789218,00.html, September 2003.

[4] British Quality Foundation, *The Model in Practice 2,* 2nd ed., London, England: British Quality Foundation, 2002, p. 86.

[5] Smith, M., "Govern IT," *British Quality Foundation IT & T Group Meeting*, London, England, January 29, 2003.

[6] Software Engineering Institute, "Capability Maturity Model®," http://www.sei.cmu.edu, July 2003.

[7] British Standards Institute, PD0026:2003, *Software and Systems Quality Framework—A Guide to the Use of ISO/IEC and Other Standards for Understanding Quality in Software and Systems*, London, England: British Standards Institute, May 2003.

[8] Gilb, T., and D. Graham, *Software Inspection,* Reading, MA: Addison-Wesley, 1993.

[9] International Standards Organization, ISO 9000:1994 and ISO 9000:2000 Quality Systems 9000.

[10] Gilb, T., "Competitive Engineering," http://www.result-planning.com/, September 2003 (Web site now replaced by http://www.gilb.com).

[11] Watkins, J., "How to Set Up and Operate a Usability Laboratory," *EuroSTAR Conference,* Edinburgh, Scotland, 2002.

Selected bibliography

Buttrick, B., "Effective Project Sponsorship—Turning the Vision into the Reality of Success," *Project Manager Today*, September–October 2003, pp. 12–13.

Information Systems Audit and Control Association, http://www.isaca.org.

IT Governance Institute, http://www.itgi.org/ITGI.

Kaplan, R. S., and D. P. Norton, *The Balanced Scorecard*, Boston, MA: Harvard Business School Press, 1996.

The National Strategy to Secure Cyberspace, http://www.whitehouse.gov/pcipb/.

Obeng, E., "Helping Stakeholders to Understand Requirements," *Project Manager Today*, July 2003, pp. 14–17.

CHAPTER

4

Roles and Quality: Managers

Contents

In this chapter I shall:

- Introduce the members of the managers' group, their roles, and activities;
- Introduce their quality viewpoint;
- Provide a framework for the managers' activities within the EFQM Excellence Model;
- Identify information flows between the managers and the other groups.

The tester challenges the project manager: "If we go live at the planned date, I predict there will be problems! I want to test the software more!" The project manager replies to the tester: "No! We go live when we planned to!"

4.1 Introducing the managers

In Chapter 3, we learned that customers want to acquire software to help them carry out their tasks. They may have a limited budget, among other constraints. They will want the software delivered as soon as possible so that they can start using it. They will need to be supported during the acquisition and use of the software.

The people who will provide the software and the support for its use are technical experts: builders, measurers, and supporters. They will focus on the details of the technical solution.

In contrast, the manager's focus is on nontechnical aspects of the work, such as budget control, planning, and reporting. Too often, the other groups see managers as the villains of the software development life cycle (SDLC). For example, our tester above will be angry that the project manager will not

delay the project to allow more testing. She may say, "Surely, the manager can see there will be problems. And, anyway, what has the project manager contributed apart from calling meetings that no one wants to go to?" But the tester may be wrong if the overall organizational imperatives are to deliver early. In this chapter, I will examine what managers do, and their value-based view of quality and why it is useful.

Managers, if they are doing their jobs well, exist to enable other people to do what is required for the organization and the customer. They are needed because IT projects and IT support are often complex, not just technically but organizationally, involving large numbers of people, a restricted budget, and a tight timescale. The technical groups (builders, measurers, and supporters) are very focused on their own activities and, naturally, value them over other people's work. They lose sight of the big picture, as well as the budget and time constraints. There will be conflicts between groups and individuals, and competition for scarce resources. Risks will be perceived differently by different groups, and, as we are finding in Chapters 2 to 7, each group holds a radically different view of quality.

So what do managers contribute? They keep everyone aligned to the goals for the customer while keeping within the time and budget constraints. They will negotiate between parties to arrange sharing of scarce resources, make decisions about how to proceed in the face of risks, listen to complaints from all directions, and soothe, calm, coach, and motivate the team. Yes, managers make a contribution. A good manager will hold the team together, through thick and thin, to achieve the goal.

4.2 Who could be in this group?

Managers include all the people who control the planning and management of software delivery and support. This will include specialist project managers and team leaders, as well people who manage departments or teams that provide ongoing services and support (see IT Managers in Chapter 2, Table 2.1). For a particular project, there may be a hierarchy of control, starting from a project/program board. Reporting to the project board might be a program manager, one or more project managers and, working under them, project leaders and team leaders. There may also be a project office providing project management support. These people are stakeholders for quality because they are responsible for the time and budget control of the project, and for meeting planned delivery requirements. These are important aspects of quality: the customer needs a product they can afford.

Of course, we all need to manage our own work. We need to plan what we do, and manage it so, in that respect, we are all managers—it is the size of what we manage that changes. For many of us, making the move from managing our own time to managing our team's time is a very difficult transition and we will look at that in this chapter.

4.3 Quality viewpoint

Managers tend to focus on two key measures: time and money. These two constraints on any work are very important. If the customer spends beyond their budget or if the software arrives too late to be of use, however technically excellent it is, we have not delivered a quality solution to the customer's problem. Therefore, this group favors the value-based definition of quality.

As we saw in Chapter 1, the value-based definition is focused on things that impact on the running of the business as a whole. Software quality is determined by a trade-off between time, effort, and cost aspects. Managers will measure the cost–benefit ratio and return on investment (ROI). They will also plan and manage within a budget for time, effort, and cost. Because of their need to contain the work within a set budget, they may also support one aspect of the manufacturing-based views of quality, that is, delivery to an agreed specification. They will see any change to the agreed specification as "scope slip" and will want to avoid that. This is very important, particularly for teams delivering products or services within a fixed-price contract or to an immovable deadline. If a change in scope means additional work, the supplying organization may take a loss. In the long term, this could mean the collapse of the organization. Managers will support processes and activities that provide an immediate and a long-term ROI: a win now for the team and the customer, and long-term customer satisfaction leading to repeat business.

Although this group may hold a view of quality which focuses on value for money, they also hold transcendent quality views. We saw in Chapter 1 that we all "know quality when we see it"; our knowledge is based on our experiences, taste, affections, loyalties, and emotions. For managers, this may mean they have a strongly held view about excellence based on their previous experience; this will vary depending on whether their premanagement experience was as a customer, a builder, a measurer, or a supporter. The "taken for granted" assumptions of managers may align particularly with that one group. Managers find themselves wrestling with the quality views of all the groups in conflict during negotiations. Unfortunately, as different people will have different assumptions, a manager, when negotiating, needs to be particularly careful not to assume that one group is correct and the others are wrong.

A common cliché about managers is that they will deliver anything provided they meet the budget and timescale, and they are not interested in quality, just in meeting the plan. When we examine behavior patterns, we can see that the managers' drive to meet the plan is essential. In a particular training course that I present, groups carry out a series of six risk assessment tasks within a set time. What I have noticed is that when a group of project managers carry out these tasks, they divide their time evenly between the six tasks. However, a group of testers or developers given the same tasks and timescale may only complete the first one or two tasks, although they will have discussed and documented the risks in much more detail. The testers

and developers did not see "missing the deadline" as a risk in itself, and, in fact, asked for more time; they "overengineered" their tasks. The project managers factored in "not meeting tight deadline" as one of the risks they needed to manage and produced a much sketchier, but adequate, solution to the tasks. This is an object lesson in why we need managers. As we will see in Chapters 5 and 6, when left to themselves, builders and measurers may pursue excellence at the expense of all other considerations. When managers negotiate with the other groups, they need to express what they require in a way that focuses on the other groups' quality views. We can see in Table 4.1 how a project manager could use the other groups' quality viewpoints to start putting across ideas about cost–benefit ratio and value.

Managers of IT projects are often recruited from the builders' group, but may also come from the customers, measurers, or supporters' groups. The group managers come from will tend to bias their quality viewpoint, but one of the jobs of the manager is to negotiate a balance between the viewpoints that is suitable for a particular situation.

4.4 Quality framework using the EFQM Excellence Model

4.4.1 The EFQM Excellence Model and the manager

There are qualifications, standards and methodologies for project management and for management processes in general, but I am not going to describe those here. I am not going to recommend a particular project management methodology. What I want to do is pick out a few important aspects of quality that are affected by management choices, and discuss these in the context of the EFQM Excellence Model we looked at in Chapter 1. We will

Table 4.1 A Project Manager Considering Other Viewpoints for Quality Impacts

Group and Quality View	*Possible Routes In*	*Example Questions*
Customer User quality view	Customer's performance measures and indicators of success	What are the financial and nonfinancial goals for you on this project? What strategic benefits do you need to realize from this project?
Supporter User and product quality view	Supporters' service-level measures	What SLAs do you have with the business? What support budget do you have for supporting the products from this project?
Builder Product and manufacturing quality view	Builder's knowledge of technical solutions	The customer needs to increase time to market for new products without compromising the excellence of service offered; what technical trade-offs do we need to make to support this? Do you know a technique or tool that would help with this?
Measurer Product and manufacturing quality view	Measurer's knowledge of quality processes	The customer needs to decrease time to market for new products without compromising the excellence of service offered; what quality processes do we need to apply to check the proposed solution? Given the limited budget, what are your recommendations for focusing testing and inspection processes?

see that we each need to tailor our methods in response to aspects of quality, culture, standards, risks, and expectations that affect our circumstances.

Whichever framework managers choose to follow, it should include some quality-management activities. The quality-management system for an organization provides a framework of policies, standards, processes, and procedures. Depending on the organization, these may be prescriptive and mandatory, perhaps because of external requirements, or they might be a framework of possible standards from which a manager chooses a suitable process for a particular project. This latter approach allows a manager to tailor the approach for a project to meet particular risks and customer expectations. I have seen it used successfully in a number of organizations. We will see an example in Chapter 6, where I will look at auditing. This selection of an appropriate process and controls is a quality-planning activity and should take place as part of project planning. The plan should include not just the schedule for the project, but also an assessment of risks and constraints, a statement of the aims and objectives for the project, and a series of planning documents describing the quality activities for the project (Figure 4.1).

This planning process will start during start-up (Chapter 9) and be refined at the start of the SDLC. However, it will continue during the SDLC and delivery (Chapters 10 and 11). Circumstances, risks, and what is important to the customer will change continuously and the management framework must reflect the need for managing constant change. Of course, managers working to fixed-price contracts will have a difficult time with this; they will either need to confine change to what can be achieved within the fixed price, agree with the customer that the fixed price is for a small, stable delivery, or renegotiate the contract. Some SDLC models, as we will see in Chapter 10, provide for managing change better than others do.

Quality planning sets the processes for this specific project for quality assurance and quality-control assessment and measurement activities, as well as setting the methods to be used for building the products. Quality assurance (QA) activities check that the processes are suitable and are being followed. Quality control (QC) checks the products for defects

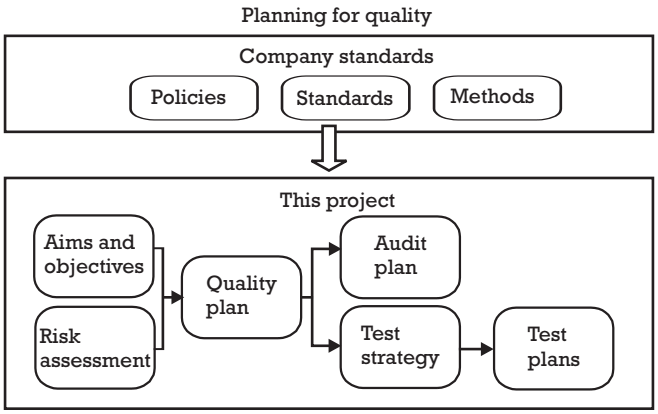


Figure 4.1 Planning for quality.

and improvements. These activities may provide feedback suggestions for improving processes and products. Figure 4.2 is an adaptation for planning that I have made of Deming's Plan–Do–Review–Improve cycle [1]. We will examine that again in Chapter 12.

If this evaluation is left until after delivery, it will be too late to learn and improve the product we are delivering, so it makes sense to build evaluation into the process steps. One of the key points about an evolutionary delivery, for example, is that it is accomplished in many small increments with a feedback loop in each increment, allowing continuous improvement and learning (see Chapter 10) [2].

In the EFQM Excellence Model [3], information systems come under the heading "partnerships and resources"—information and information systems are resources that enable the organization to achieve its strategic goals. If we look at an organizational measurement system, such as the Balanced Scorecard, information systems are seen as a critically important tool to enable employees to work effectively, so that they may contribute to the goals for the organization. We will look at how managers use the Balanced Scorecard later in this chapter.

Managers need to be aware of the customer's strategy and quality framework, including any measurements and targets, in order to deliver services and products that help the customers meet their organizations' quality targets. The emphasis will change, and this will change what constitutes quality for the organization. The organization may be driven by a need to meet a particular external standard, to keep within legislative or regulatory bounds, to increase market share, to reduce time to market, or to be a world leader for excellence of service. The manager's ability to recognize and adapt to the customer's quality framework is a critical success factor in the effectiveness of their relationship. The manager should refer to the customer's goals when setting team and project goals, translating these into personal objectives and targets for themselves, and the builders, measurers, and supporters. This quality framework will drive what the customer needs from the software.

In Chapter 1, we looked at the EFQM Excellence Model and how it is divided into nine parts; five enabling criteria and four criteria for measuring results. In Section 4.4.2 we will look at how the EFQM Excellence Model

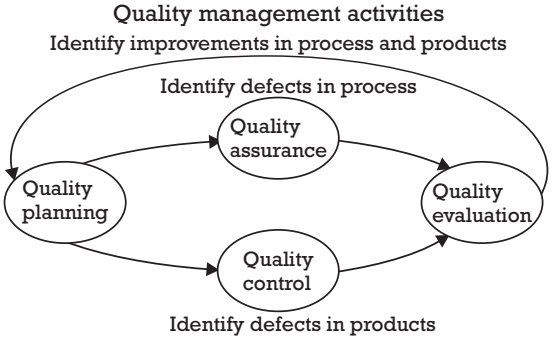


Figure 4.2 A quality-management planning cycle.

enablers could be interpreted for managers, and in Section 4.4.3 we will look at manager results. Remember that this model is based on the Fundamental Concepts of Excellence we discussed in Chapter 1, and that equivalent models such as the Baldrige model are available.

4.4.2 EFQM Excellence Model enablers for the managers

4.4.2.1 Leadership

Managers provide leadership for their teams. In order to provide leadership, they need leadership from their own managers and the board. The manager will follow the lead given by the organization in matters of quality viewpoint and, hence, on whether value, fitness for purpose, manufacturing defects, or product attributes will be the leading quality viewpoint. Managers may also be torn between conflicting leads from the customer organization and more senior management in their own organization.

The manager is often the main conduit between the customer and the rest of the team, relaying the customer's aims and objectives. Through leadership, the effective manager communicates these to the team in a way that creates a common vision and unity of purpose. The CMMI® framework that we discussed in Chapter 1 includes a specific management practice to "Establish a Shared Vision" [4] for the team. A colleague commented to me:

> This vital "big picture" increases commitment and enables all members of the team to think beyond their own specialism and use alignment to the customer's objectives to validate their activities. Asking a selection of team members to describe the objectives of one project resulted in a variety of incorrect answers. The reason was that the managers believed in communicating the minimum information that they thought the practitioners needed to carry out their tasks. Unsurprisingly, commitment in the team was low but it was also evident that a lot of activity in the team did not align with the project's objectives, and would have been challenged if the team were aware of the project's purpose.

The ability of a manager to inspire and motivate the team through leadership cannot be underestimated. The behavior of someone the team accepts as a leader is characterized by honesty, trustworthiness, the ability to keep a confidence, respect for others, and enthusiasm for achieving the aims and objectives. If the leader does not believe in the cause, no one else will. I remember visiting one project with some colleagues to provide consultancy. The project had been running for over a year and the team was tackling some extremely difficult but interesting problems. When we said, "It's difficult but we can do it—and it'll be fun!" a team member said to me that none of the management team leading the project to that point had indicated that achieving the project's aims was anything but difficult and unpleasant. If the managers were not positive, how could the team be? When the new management took a more robust attitude to the team's ability to solve

problems, the project started to succeed. Of course, if managers lie about the difficulty, they will quickly lose their team's motivation, but part of a leader's role is to understand what is necessary and what is possible.

4.4.2.2 Policy and Strategy

Managers set policy and strategy for their own areas based on their organization's policy and strategy. This will include policy and strategy for QA and QC, but we also need to define how the build and support activities are done. In Chapters 5, 6, and 7, we will see some aspects of policy and strategy for builders, measurers, and supporters; what managers need to consider is how the customer policy and strategy are reflected in the strategy for the project and, hence, for the all the work done.

Policies are brief statements that are the rules for how an organization conducts itself. Managers also need policies for management activities; these may be documented or unwritten ("We always do it this way."). The decision about whether or not to document policy depends on the size of the organization. (In this case, for example, does a particular project require its own policy document, or does everyone in the project "know the rules"? Is the policy statement at the organization level sufficient?) Policy statements for management might include topics such as:

- Authorization, for example, "documents may not be released to customers without an authorization sign off by a team leader."
- Escalation, for example, "the project manager will always be informed of risks, issues and defects identified with an impact level of 4 or above."
- Reporting, for example, "test team leaders will report progress against plans to the test manager daily at 8:30 a.m."

Strategies for managing show how the policies will be applied and what approach will be taken in a particular case. One mistake I have seen inexperienced managers make is to believe that the project plan is just a schedule. In fact, the schedule is just part of what is needed to plan and control a project. Appendix B shows some of the documents we might need, including a quality plan, a configuration management plan, and a risk management plan.

On a small piece of work, these might be used as a checklist of things to think about, or a single document might be written that combines them all. In that case, keep it small—the size of the plan is not an indicator of its quality. One very good aspect of tailoring is to allow the manager to combine and abbreviate documents where this is appropriate.

On a major project, we might need a whole set of documents, each of which describes some aspect of the overall strategy for the work. Each document may have a different author, and part of the manager's job, with the team, is to make sure that the documents do make up a document family, do complement rather than contradict each other, there are no gaps, and the overlaps are minimized. Part of the reason for defining a document

family is to allow child documents to just document differences and additional information. A common error in large projects is to become overwhelmed with paperwork and electronic documents. Keep repetition across documents to a minimum, as it saves reading and update time.

If you are following a published project management method, you will find that there will be a set of documents defined for your use. It is worth looking at the rules in the method to see whether it is allowable to tailor, combine, and minimize documents.

4.4.2.3 People

Managers spend much of their time dealing with people. They are often the hub of communication between the customer and the other groups, and they also deal with communications up and down any hierarchies. Managers need to be good communicators, and this means that they need to:

- Be good at receiving information and showing they have received it. They must read and listen well, and provide feedback.
- Be good at giving information and making sure people have received it. They must speak and write clearly and elicit feedback.
- Understand other people and treat them fairly. They must persuade, negotiate, empathize, and know when to compromise and when to be firm, treating everyone fairly and taking into account their different communication needs and quality viewpoints.

We have seen in Chapter 3 that one of reasons customers and builders sometimes do not get on is because of a difference in communication styles, and this is true for people in general. In Chapters 8 to 12, as we look at the stages of work that need to be managed, I will mention some techniques that help us improve communication.

Managers have a difficult mix of skills to acquire, and not all managers have all these skills (which of us has?). One trick I find useful is to use a "personal improvement cycle." As we saw in Section 4.4.1 and will see in Chapter 12 when we look at evaluating software development processes, we use a plan–do–review–improve cycle. On a personal level it simply means reflecting on how we manage ourselves during communication with others and what we could do to improve the communication, not just for ourselves, but also for the other people. Particular points to consider are cultural differences, personality differences, and communication styles, particularly when giving and receiving criticism [5–7].

I have noticed that one key factor in the success of a manager is consistency of behavior; if people know they can trust you, they work with you. I remember one project manager who was very hard on everyone, but this was less of a problem for us "down the hierarchy" when we realized that he gave the same grief "up the hierarchy" to the project board and executives.

Another key factor is not being remote. Another very good project manager helped during a crisis weekend when technical staff were working

extra hours on some of the menial and housekeeping tasks to free time for the team. This effort was much appreciated; he was seen to be in and working when the team was doing overtime, and the work he was doing at that point contributed directly to removing some of their burden.

4.4.2.4 Partnerships and Resources

I have worked in situations where the customer chose to bring in specialists from a variety of organizations. For example, on one project, managers came from the customer organization and were also brought in from a specialist project-management consultancy; the builders and measurers came from third-party organizations as well as from the in-house IT group; and the supporters were a separate group from the IT group, reporting through a different management structure, although they were in-house. The project manager spent much time negotiating between these parties. All were partners in the project, and were controlling different resources that were required if the project was to be successful.

An IT manager has pointed out to me that one key skill that a good manager has is political astuteness*:*

> Managers often lead a virtual team that is united only by a shared task. Whereas team leaders manage a group of practitioners in their own discipline (such as builders or measurers), senior managers lead a matrix of specialists drawn from various functional areas (or companies), whose work they do not understand in depth. These managers cannot depend on traditional sources of power such as the line hierarchy or resource ownership; instead, they deploy a range of influencing tactics. They operate in two worlds simultaneously: the rational world of budgets and schedules and the shadowy world of organizational politics, which the specialists normally don't see.

Some new managers do all the things that I have recommended elsewhere in this chapter and still find unexpected opponents blocking or disrupting their projects. The same IT manager continued:

> The experienced manager who spends all day in meetings and every evening having dinner with key project stakeholders may be a joke in the team but always gets the best people and delivers the right product.

These organizational partnerships through influence are critical to success, and the ability to nourish them makes the difference between those of us who pursue technical careers and those of us who follow organizational careers. The ability to influence is not a sign of dishonesty, despite the suspicion of some IT technical staff; it is a way of communicating with people who are stakeholders and have different communication styles (see Appendix A for communication styles).

4.4.2.5 Processes

The processes used by managers include skills such as planning, estimating and reporting, for which there are various project management methods and tools available. What I will do is to pick out some aspects of the manager's processes that particularly affect teamwork and quality.

Learning to be a manager. One aspect of IT management that I have observed over the years is that people are given their first management responsibilities without having the role explained to them or their existing work removed; it is as though we believe that management activities are intuitive and take no time. In one typical instance, one of the development team, let us call him Jeff Teamleader, was asked to team lead part way through a major project. A few weeks later when I visited the project, Jeff came to me in a state of some distress; he was unable to carry out any of his work and was falling further and further behind. He could not understand what had happened.

When we talked it through, it turned out that until Jeff had been given the team leader role, he had a technically complex but organizationally simple role; he had never needed a to-do list or a plan of action. Now, as team leader, he had to plan and coordinate what the team as a whole was doing, produce some management deliverables such as estimates and reports, go to meetings with the project manager, and negotiate and deal with conflict inside the team and with other teams. Also, he still had his own technical work to do. He had simply become overwhelmed and unable to do anything as a consequence.

What was required? He needed a simple personal plan and a simple team plan, that would enable him with his team to make the big delivery in a few weeks time, by breaking it down into chunks. Every day, he needed to know what he and the team had to do that day, in order to deliver what was needed in the current week, so that the longer plan would work.

First, we sorted out his personal plan. We made a list of everything that he had been asked to do or thought he had to do, including all the nontechnical and nonproject tasks and building a plan for the team. We scored each item on the to-do list with an urgency, importance and size score. We quickly identified some things that he did not have to do. Then we divided up the to-do list as in Table 4.2, by deadline, importance, and urgency, putting together a day-by-day plan for the coming week, including a 5-minute progress review at the end of each day and a 10-minute session to plan the following week at the end of Friday.

Once Jeff had this simple plan for next few days, he could start to plan the team's work in the same way, and sort which parts of his own technical role to keep and which to delegate. It was much clearer to him and to them what really needed doing day to day. He calmed down; he could get a grip on the situation. Why hadn't his manager, Liz, explained this to him as he took over? I found two reasons when I talked to her:

Table 4.2 Simple Personal or Team Plan Layout

This week	*Morning*	*Afternoon*	
Monday			
Tuesday			
Wednesday			
Thursday			
Friday			
Next week start:			
Next week complete:			
This month complete:			
Goal: by (date) have completed (deliverable)			
To-do list			
Item	Importance	Urgency/deadline	Size

- Liz was dealing with larger and more complex problems, using project management tools and methods that would not be appropriate. Her attitude was that "Jeff Teamleader doesn't need that yet."
- Liz now took simple planning for granted and had forgotten that she too had struggled with her first team leader role. She said, "Well, I'm disappointed if he didn't realize that he needed to plan his time; I'd have thought it was obvious that he needed to...."

The problem that Jeff Teamleader then had was that his plan was disrupted by other people interrupting him and the team with questions and other meetings. A plan will always change; we plan partly to have an idea of what we need to do when, but also so that when things change we can deal with the change more easily. Ideas I have found useful are:

- Knowing your highest priorities: the things that you cannot compromise if the team is to meet its goal. That means you know which tasks you can delay or even cancel.
- Setting aside time that cannot be interrupted for particular individuals and for the team. This is time when people concentrate on the most important and most difficult tasks. You can achieve this by making appointments with yourself.
- Not making the plan too perfect; it just needs to be good enough to allow you to understand progress and changes.

Planning and estimating for quality activities. As part of planning, we need estimates of how long things will take to do. Estimates are really just guesses with more or less certainty behind them. As we saw in Chapter 3, the customers need to understand how precise and accurate our estimates are, so it is important to provide them as a range of times, costs, and resource usage, becoming more precise as we gain certainty and understanding. The teamwork aspects of estimating are:

- Listening to your team—they know how long different tasks will take and what resources are needed
- Listening to your customer and manager—what constraints are there that affect the estimate? For example, if the overall budget cannot exceed a certain amount, then your estimation task is to show what can be delivered for that price.
- Explaining to the customer the basis for the estimates—the assumptions made, the gaps, and the precision and accuracy of the estimates

There is a tendency to underestimate how long QA and QC tasks take, and to forget to estimate for changes and rework following the QA or QC. Table 4.3 suggests some tasks that you may need to include in your estimates, and to get the details, as a manager you would need to consult your

Table 4.3 QA and QC Tasks to Include in Plans and Estimates

Drivers for percentage of QA/QC activities to cost of project	Aims and objectives for the project/business, risks to be contained, constraints, how important are QA and QC activities for this project?
	All the deliverables for the project, their importance and risk associated with faults, what QA/QC does each need?
	Acceptance criteria for the project as a whole, Acceptance criteria for each deliverable, which QA/QC activities can measure against the acceptance criteria, and completion criteria for those activities
QA activities—audit and process review	Allow time, cost, resource usage for internal and external audit for:
	Planning
	Preparation (times number of auditors)
	Audit activities, for example, meetings (times number of auditors and interviewees)
	Follow-up and rework (time spent by auditor and auditee)
	Evaluation and process improvement (may be at audit end, increment/phase end, project end)
QC activities—document review	Allow time, cost, resource usage for:
	Planning, including selection of what is to be reviewed and the method to be used
	Preparation (times number of reviewers)
	Meeting (times number of reviewers)
	Follow-up and rework
QC activities—testing	Allow time, cost, resource usage for:
	Planning, including deciding on the level and types of testing, setup of management and control processes
	Acquisition—include test design, data, environment, tools
	Measurement—include execution, reporting
	Follow-up—include rework and rerun of tests

QA and QC experts to understand which techniques would be useful for your project. You also need to estimate what knowledge and resources you will require, and when you will need them. Tools, environments, and special skills may not be available to your team all the time, and there may be conflict with other teams for use of these resources, or they may not be available in your organization. In that case, you may need to estimate for outsourcing or purchasing resources.

Monitoring, control, and measurement of progress. Once we have our plan and our estimates, we need to track progress and make changes to the plan as necessary. If, like Jeff Teamleader, our team's work is contributing to a part of the overall plan, we need to be careful how we manage changes, as slippage in our work may have an effect on other people's work; there are dependencies between the tasks. In a large project, the overall project schedule may be controlled on a schedule of activities and tasks which shows these dependencies on a PERT (program evaluation review technique) chart. A PERT chart shows the length of time each task takes and the dependencies between tasks; it is a useful planning technique. A simple example is shown in Figure 4.3.

You can see from the figure that some tasks have to complete before others can start, but some can take place in parallel. You will see on each task the task name, the duration of the task, the earliest and latest days the task could start and finish, and the *slack* on the task. The slack is the amount of time you can let a task slip before it makes the subsequent, dependent tasks slip. The path through the tasks with zero slack is the *critical path,* shown in bold in the figure. If anything on this path is late, the whole project will be late. If finishing on time is important, planning using a PERT chart to

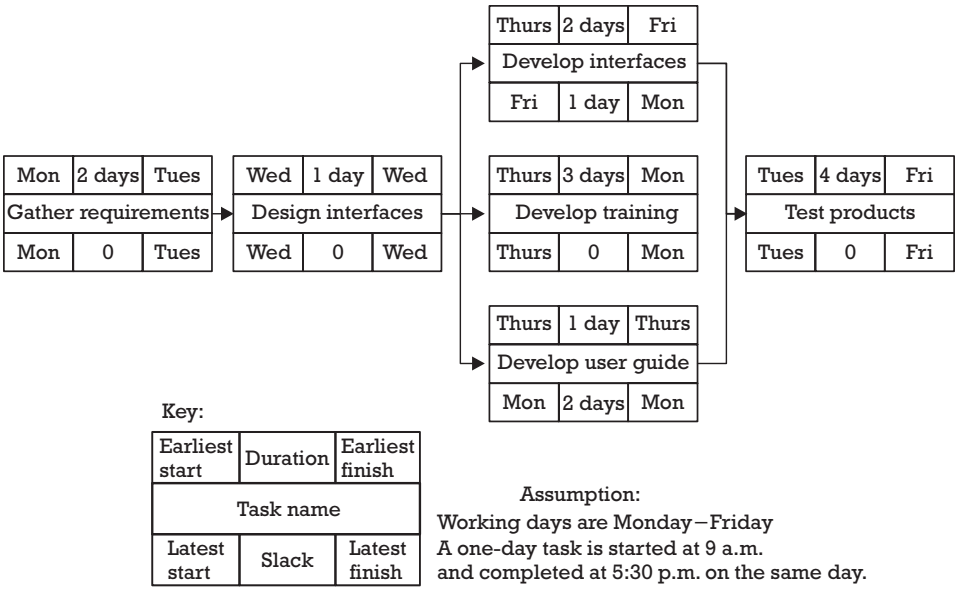


Figure 4.3 Simple PERT chart.

understand dependencies and slack can be very useful. When monitoring, small amounts of slippage within slack may not be important to us, but if the slippage means that later tasks on the critical path will not complete on time, it means we will not meet the deadline for the project. We might report using on a red–amber–green traffic lights system:

- Green tasks are completing within schedule.
- Amber tasks are finishing within slack.
- Red tasks have used up slack.

We can then identify where we have time problems: "If it weren't for task X, we'd be a green project." Different levels of management may require different levels of detail in reporting. I have seen some organizations concentrate on reporting on red and amber tasks only, whereas others only report to the project board if the slip exceeds a certain amount. This can cause problems with accumulated small slips. For example, if managers only track slippage by monitoring individual slips above a certain number of days, but there have been many small slips, they may not realize that the real hidden slippage in the project may be much larger than documented.

There are other things to track apart from time used and time to complete against estimated time. For example:

- Overall budget to date against spending to date.
- Number of deliverables completed against how many you expected to complete by now.
- Cost to produce deliverables to date compared with what we expected to spend on those deliverables.
- Quality of the deliverables against their acceptance criteria.
- Morale of the team (all groups—how happy are they now?).
- Delivery against targets.
- The organization's objectives and the customers/supporters' requirements and how these are changing; how the project is changing to meet the changing requirements.

We will talk in Chapter 10 about changing requirements during the software development life cycle, but here I want to discuss the use of a tracking and reporting mechanism that is used at the organizational level and could be adapted for an IT project: the Balanced Scorecard and its variants, sometimes referred to as the project dashboard, which we will look at below. Another mechanism, Earned Value, is discussed in Appendix A.

4.4.3 EFQM Excellence Model results for the managers

4.4.3.1 Customer Results

The customer results for the manager to track are:

- Customer perception of the other groups. Managers should meet their customers regularly, and as part of such meetings need to check how the customer feels about things. I remember a salesman once saying to me, "What do you think you deliver?" I came up with a number of answers—completed audits and tests, reports, tested software, and so on. He said, "No—what you are delivering is happiness. Is the customer happy to see you? Are they pleased you are there?" We can measure this right now, throughout our work, and we can adapt to deal with any problems
- Customer performance results are also important; for example, do we get repeat business or extended projects from the customer? Have we a good reputation as a team and individually as managers? These measures may not be apparent until too late if we have not worked on customer perceptions during the project.

4.4.3.2 People Results

Here, we consider the measures we might apply to managers themselves. Table 4.4 suggests some measures. The first of these are perceptions—how the managers feel about themselves in relation to the other groups and their work. The second group are the "harder" performance measures, staff turnover, for example.

4.4.3.3 Society Results

Managers have a role in ensuring that society as a whole is considered, in line with corporate objectives. Table 4.5 shows some example results for measurement. Quality here is not just the quality of the product itself, but the quality of its effect on the organization and on society as a whole. Managers will have a responsibility for ensuring that their own team's work does not breach standards in areas such as environmental management and impact, organizational security, and corporate governance.

Table 4.4 People Results—Managers

	Example of Possible Measures
Perceptions—surveys, interviews, compliments, and complaints	Has a motivational study provided positive feedback?
	Are managers satisfied with their career paths, rewards?
	Are the teams satisfied with their managers? Is the level of management and control right for the individuals in the team?
Performance indicators—internal to managers group	Does the managers group match up with the required qualifications and competencies? For example, do they have appropriate BCS qualifications [8]?
	Are there particular signals of stress among the managers: excessive hours, absenteeism, and sickness levels compared with other groups?
	What is the level of staff turnover compared with other groups and do people want to be recruited into this group? For example, if we advertise a team leader role internally, do people apply?

Table 4.5 Society Results—How the Managers Relate to Society

	Example of Possible Measures
Perceptions—external to the group: surveys, interviews, compliments, and complaints	Are the managers seen by the organization and wider society as acting ethically?
	Do the managers ensure that the processes and systems act properly with regard to health risks, safety, hazards, and the environment?
Performance indicators—internal to managers group	Has the manager encouraged the team to work toward accolades, for example, for paper- and energy-saving initiatives, in line with corporate initiatives?
	Have the managers supported the measurement group in checking compliance with external authorities' certification?
	Has the manager ensured that the team's work complies to standards such as the ISO 14000 family (Environmental Management) and the ISO 17799 family (Security Management)?

4.4.3.4 Key Performance Results

It is important that the manager choose an appropriate set of measures for monitoring and controlling the project. This may seem obvious, but often we choose measures that are not appropriate—we make the mistake of measuring what we have always measured, or what is easy to measure, rather than what we need to know.

Within the organization as a whole, there will be a strategy with associated measures, and both commercial and nonprofit organizations are starting to use Kaplan and Norton's Balanced Scorecard [9–12]. This is a method to balance different types of measure against each other. Examples might be:

- Long term versus short term;
- Financial versus nonfinancial;
- Leading and lagging indicators;
- Internal and external perspectives;
- Objective and subjective perspectives.

A simple example might be a balanced scorecard for a journey: We want to get there as fast as possible, which will be expensive, but we also want to travel as cheaply as possible, which may mean a slower journey to conserve fuel. There is a trade-off or balance between speed and cost.

Kaplan and Norton's scorecard typically looks something like Figure 4.4. Financial, customer, internal, and learning measures are balanced against each other, with balanced objectives and targets.

There are as many variations in scorecards as there are organizations using them; [9–12] present several examples from real companies. What I suggest is that if we know the goal and strategy for the organization, we can build a balanced scorecard for measuring quality in an IT project by balancing the quality views against each other. Figure 4.5 shows an example.

The measures in this scorecard can:

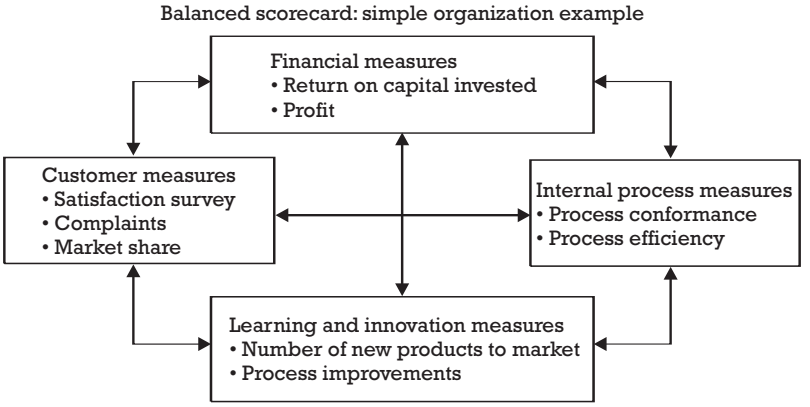


Figure 4.4 Simplified typical balanced scorecard. (*After*: [9], p. 9.)

- Reflect the organization's strategic measures and ensure that those on the team all understand those measures;
- Balance the conflict between the value, user, product, and manufacturing views of quality;
- Be used continuously to report progress, not just against time and deliverables, but against quality targets;
- Be kept on one page or screen so it is easy to take in and agree upon;
- Be backed up by other measures and detailed explanation *if required* but does speak for itself.

4.5 Communication between the managers and other groups

4.5.1 Managers and communication cycles

We will see in Chapters 8 to 12 that managers are involved in the whole life span of a piece of software, from its conception, through the software development life cycle (SDLC), during delivery, and postdelivery until decommissioning.

Managers need to negotiate with others and also to control project progress. This can mean that the communication between managers and others can be difficult. Suppose that Liz Manager is waiting for a progress report from Jeff Teamleader. The report was due yesterday and she has heard nothing. How should she approach Jeff? Maybe Liz is not good at confronting people. She might be tempted to wait and see what happens. But as manager, she needs to know why the report is late; there may be a problem that she needs to resolve that Jeff is reluctant to tell her about. One manager I worked for was quite unable to ask people for late reports. He would put a notice on the project notice board: "Will all team leaders please ensure that their reports are delivered on time; the following have not yet delivered this week's report…." The end result of his being unable to speak to his team

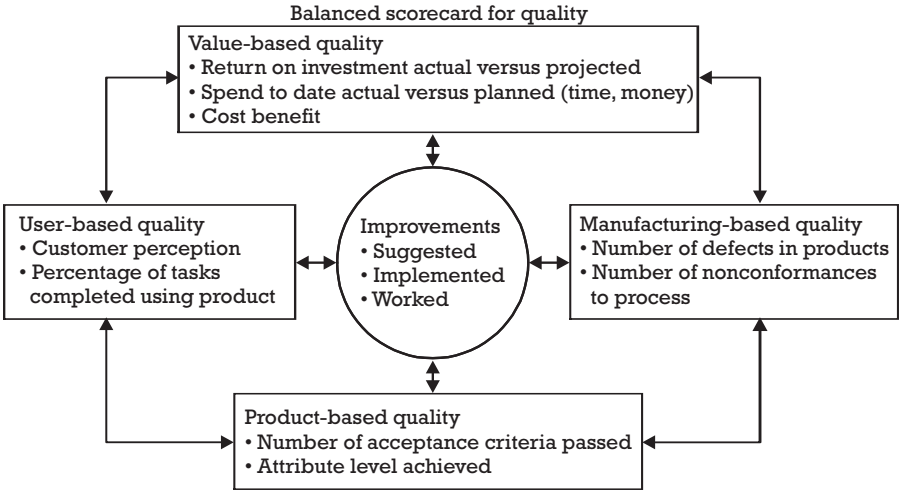


Figure 4.5 Quality balanced scorecard. (*After*: [9], p. 9.)

leaders directly to ask for their reports was that the team leaders, who did not like the manager, did not understand the use of the reports, and who by and large were more experienced, started to put more energy into their "being late with the report" game. One particular old-timer said to me, "I always write the report on time, but I'm not going to give it to him unless he asks me himself; its an insult the way he puts those notices on the board without speaking to me, when I sit right next to it." I was too junior to ask the manager for his thoughts, but my reading of his behavior was he was frightened of the team leaders and so could not give or receive criticism successfully. All he needed to do was go to the team leaders and ask for the reports. I suspect that if he had done this once, they would have stopped the game. During complicated interactions, we can only alter our own behavior, and hope that if we behave well, others will be encouraged to behave well [5–7, 13]. So, Liz Manager needs to ask Jeff Teamleader for his report, and she needs to do that in a direct way. What affects Jeff's response to the request for the report is the manner in which it is made. Writers like Wagner [7] who use transactional analysis to describe behavior say we are several people at once—we have an internal dialogue that informs the dialogue we have with the external world. We can choose to behave like adults and treat others as adults, or we can behave like children or like parents. The ego states are:

- Adult—deals in facts: "The report was due yesterday."
- Nurturing parent—wants to help: "I see you are having problems with the report."
- Critical parent—tells people off: "Why are you always late with the report?"
- Natural child—likes to play and have fun: "Let's have lunch and think about it afterwards!"

- Rebellious child—does the opposite of what others want: "I'm not doing the report!"
- Compliant child—assumes the blame: "Oh, I'm hopeless. I'm late again."

Adult-to-adult transactions result in facts being communicated, but if you come and speak to me as though you are my parent, and are critical ("Why are you always late with the report?"), I may respond as a rebellious child ("I'm not doing the stupid report. It's a waste of time. You do it.") This is called a crossed communication; the unhelpful ego states feed off each other and we build up to an argument. We could have had a more constructive, uncrossed dialogue:

Adult: "Is the report ready?"
Adult: "No, but I will finish it this afternoon."

The question is couched neutrally as a request for facts and so is responded to with a neutral fact. Remember also that tone of voice and body language will also speak volumes; if the questioner uses "adult" wording but a "critical parent" tone of voice, she/he is still a "critical parent."

Figure 4.6 shows how we can build up an expectation of poor communication with those around us; our anticipation of problems causes us to communicate in a way that encourages a problematic response and it turns into a vicious circle. We give out the wrong signals and we get the wrong signals back.

4.5.2 The reporting process

One key process for all managers is the reporting process. Measurers gather information about processes and products and the managers, including test managers and quality managers, need to pass this on to other people in a way that will be acceptable and understandable. Good information design is

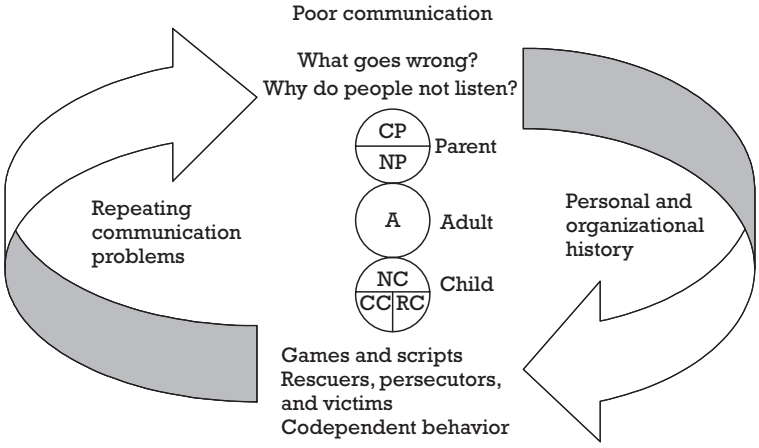


Figure 4.6 Cycles of poor communication—our audience and us. (*After*: [7].)

vital; it is easy to hide your message in your report so that it is ignored or misinterpreted. However well you have designed your audit or test process, however well you execute the process, if your reports are ignored, you may as well have not bothered doing it. Reports are the managers' deliverable to all the other groups:

- They help us understand progress and priorities.
- They help in decision making and risk assessment (they should show the bad news and good news!).
- They inform management, our colleagues, and our teams.

When deciding what to report, we need to examine the facts. First, do we have any facts? Managers need to report measurable facts, interpretation of those facts, and people's perceptions, but should distinguish between them. The measurers will provide the managers with data, which need to be interpreted and reported, for example:

- "The audit found 23 critical deviations from the configuration-management standard" is a fact.
- "The nature of the deviations increases the risk of failure in this project" is an interpretation of the fact.
- "The project team are worried that the lack of a configuration management process means that they cannot know whether they have completed the build" is a report of staff perception that comments on both the fact and the interpretation.

All three provide important information which should be reported.

When deciding what to report, do not collect information for the sake of it. One mistake many people make is reporting either on what has always been reported without questioning why, or only reporting on information that is easy to collect. Also, be prepared to report bad news as well as good news. In my experience, people want to hear the bad news; what they do not want is to get it when it is too late and without any ideas for improving the situation. Table 4.6 gives some ideas for consideration when deciding on measurement reporting.

Table 4.6 Planning Reporting—Questions

Who are our audience?	Customers, managers, builders, supporters, other measurers
Who is interested?	Internal and external bodies
	Who else?
What do they need to know?	Time, money, quality (and include the different quality viewpoints),
What are they interested in?	progress toward a goal
	What else?
When do they need it?	Daily, weekly, monthly, yearly
	When an emergency happens? As risks increase? When issues arise?
	What other cycles for your audience?
Why are they interested?	Making decisions, assessing risks, looking for improvements
	What else?

We often use graphics to report because they reveal data, especially for displaying complex ideas clearly, precisely, and efficiently but it is important to remember that graphics can be abused, either by using graphics that are not necessary or by distorting the data with the graphic [14–16]. Our eyes interpret the representation of two- and three-dimensional images as having area or volume, and if these are used to show one-dimensional data, we will see it as larger than it is, as we see in Figure 4.7.

One colleague remarked to me, "But I want to exaggerate the test results—I want the customers to be really, really worried…." I think he was joking, but the real point is, do you know that the graphs you produce are accurate and will be accurately interpreted? There are some notes and other examples in Appendix A about the measures Tufte [15] suggests for checking your graphics: the *lie factor*, the *data–ink ratio*, and the *data density* of the graphic.

Make sure you understand why you are using the graphic; is it for description, exploration, tabulation, or just decoration? Make sure you know what message you are trying to put across. We tend to overdecorate our graphics [14–16] and thus hide what we are trying to say. Edward Tufte [15] suggests that we use "the greatest number of ideas in the shortest time with the least ink in the smallest space" if we want to show the data. Many numbers in a small space encourages the eye to compare data. I strongly recommend that you look at his work to understand good and bad approaches to information design and reporting.

Managers need information from other groups in order to understand the constraints on delivering their wish list. Table 4.7 lists the information that managers need from the other groups.

Managers are conduits for information flow; they will gather information from all the groups and communicate it back to all the groups as reports, tactics, and decisions (Table 4.8 and Figure 4.8). They will do this at every stage in the life of the software.

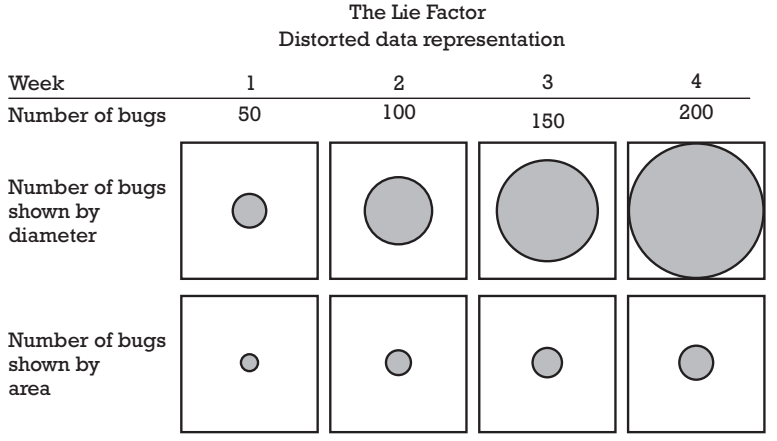


Figure 4.7 Distorted data representation. (*After:* [15].)

Table 4.7 Information That Managers Need from Others

Customers	Goals, targets, strategy for the customer
	Business constraints
	Changes to circumstances
	Acceptance criteria, set, reported on as pass or fail
Other managers	Progress
	Changes to circumstances
Builders	Progress
	Problems
	Decisions required
Supporters	Technical constraints
	Timing constraints for delivery
	Changes to circumstances
	Acceptance criteria, set, reported on as pass or fail
Measurers	Quality measures—number of nonconformances, defects, number of acceptance criteria passed and failed
	Progress
	Problems
	Decisions required

Table 4.8 Information That Managers Have That Others Need

At all stages	Information gathered from each group
	Experiences from previous projects
	Constraints and policies
	Effect of changes
	Tactics and decisions based on policy, strategy, and circumstances

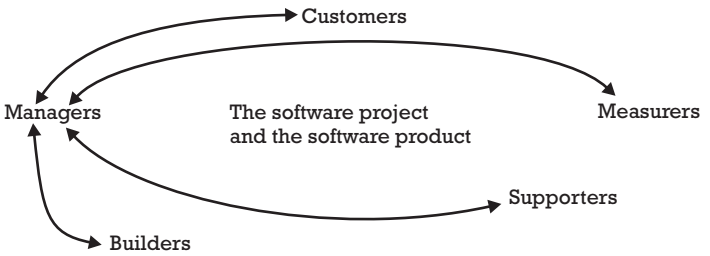


Figure 4.8 Communication between groups.

4.6 Summary of the group

Managers must communicate well with the other groups if they are to be effective. They are responsible for planning, estimating, and controlling work, as well as for making and enacting decisions. These are not intuitive skills, and new managers need help as they take on the role.

Managers are the conduit for information flow through and between the groups. Managers need to negotiate between the groups and ensure that the optimum course is taken at each point. Use of a Balanced Scorecard will

help in this negotiation. Use of clear reports is vital; managers need to ensure that their reports tell the story they are intended to tell.

Managers are sometimes put in the position of being their team's "parent" and need to consider carefully how they give and receive criticism.

References

[1] The W. Edwards Deming Institute, "Deming's Teachings," http://www.deming.org/theman/articles/articles_gbnf04.html, November 2003.

[2] Gilb, T., see papers on Evolutionary Delivery on http://www.gilb.com, including "Competitive Engineering: A Handbook for Systems and Software Engineering," September 2003.

[3] European Foundation for Quality Management, "EFQM Excellence Model," http://www. efqm.org, August 2003.

[4] Software Engineering Institute, "Capability Maturity Model® Integrations (CMMI®), Version 1.1," http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tr004.pdf, October 2003.

[5] Tannen, D., "The Power of Talk," *Harvard Business Review*, September 1995, pp. 138–148.

[6] Hathaway, P., *Giving and Receiving Criticism*, Los Altos, CA: Crisp Publications, 1990.

[7] Wagner, A., *The Transactional Manager—How to Solve People Problems with Transactional Analysis*, Denver, CO: T.A. Communications, 1981.

[8] British Computer Society (BCS), "BCS Qualifications," http://www1.bcs.org.uk/link.asp?sectionID=574, September 2003.

[9] Kaplan, R. S., and D. P. Norton, *The Balanced Scorecard*, Boston, MA: Harvard Business School Press, 1996.

[10] Bourne, M., and P. Bourne, *Balanced Scorecard in a Week*, London, England: Hodder and Stoughton, 2000.

[11] Olve, N. G., and A. Sjöstrand, *The Balanced Scorecard*, Oxford, England: Capstone, 2002.

[12] British Quality Foundation and TQMI, "Using the Model and the Scorecard," *Seminar*, Kettering, United Kingdom, September 2003.

[13] Copeland, L., "Testing as Co-Dependent Behaviour," *SIGiST Conference, the BCS Specialist Interest Group in Software Testing,* London 1999.

[14] Evans, I., "Get Your Message Across!" *EuroSTAR Conference*, Edinburgh, Scotland, 2002.

[15] Tufte, E., *Visual Display of Quantitative Information*, Cheshire, CT: Graphics Press, 1983.

[16] Huff, D., *How to Lie with Statistics*, New York: Penguin, 1974.

Selected bibliography

Adair, J., *Effective Teambuilding*, London, England: Pan, 1986.

Adair, J., *Effective Time Management*, London, England: Pan, 1988.

Berne, E., *Games People Play*, New York: Penguin, 1970.

Boddy, D., and D. Buchanan, *Take the Lead: Interpersonal Skills for Project Managers*, London, England: Financial Times Press, 1992.

Brooks, F. P., *The Mythical Man Month*, Reading, MA: Addison-Wesley Longman, 1995.

Burnett, K., *The Project Management Paradigm*, London, England: Springer, Practitioner Series, 1998.

Chartered Management Institute (on-line checklists about management and quality topics), http://www.managers.org.uk.

Clarkson, M., *Developing IT Staff*, London, England: Springer, Practitioner Series, 2001.

Crosby, P., *Quality Is Free*, New York: Mentor, 1980.

de Bono, E., *Six Thinking Hats®*, New York: Penguin, 2000.

Evans, I., and I. Macfarlane, *A Dictionary of IT Service Management*, ITSMF, 2001.

Grove Consultants, "What Test Managers Say ... and Why Project Managers Don't Listen," play presented at EuroSTAR Conference, Stockholm, Sweden, 2001.

Hadley, T., "More Than a Thank You," *UK Excellence*, August/September 2003.

Handy, C., *Understanding Organizations*, New York: Penguin, 1993.

Macfarlane, I., and C. Rudd, *IT Service Management*, Reading, England: IT Service Management Forum, 2001.

Mullins, L. J., *Management and Organisational Behaviour*, 5th ed., New York: Financial Times/Pitman, 1999.

Pas, J., "Emotional Intelligence," *EuroSTAR Workshop*, Stockholm, Sweden, 2001.

Project Management Today (monthly journal), http://www.pmtoday.co.uk.

Schein, E., *Organizational Culture and Leadership*, San Francisco, CA: Jossey-Bass, 1997.

Smith, J., *How to Be a Better Time Manager*, London, England: Kogan Page, 1997.

CHAPTER

5

Roles and Quality: Builders

Contents

In this chapter I shall:

- Introduce the members of the builders group, their roles, and activities;
- Introduce their quality viewpoint;
- Provide a framework for the builders within the EFQM Excellence Model;
- Identify information flows between the builders and the other groups.

So I spent 2 months designing and building the new interface and its underlying software. It's got everything we discussed with the users, and I added some extras that looked useful, after I'd talked to the customer last week. You know I was here all night last Thursday getting it finished for delivery into test on Friday. And then at 10 o'clock on Friday morning, the tester comes up to me and starts moaning about the interface being too complex. How long could she have spent looking at it? I ask you! I just blew my top. I said, "It's my system; I designed it and I built it and I don't care what you think of it! It's not your business to criticize what it does! You weren't there when we discussed the spec." We do all the work and they do nothing but moan!

—Software engineer having a dramatic moment when the tester complained (rather rudely) about the complexity of the software interface

5.1 Introducing the builders

Of course, this group is vital to software quality—not only do they build the products, but they build in quality. The people in this group are wrestling with difficult problems as they create

new products. Why is our software engineer so angry about the software being criticized by the tester? There are a number of reasons:

- Software engineers pride themselves in doing a good job; they want to build excellent products with few defects. In this case, the engineer had done groundbreaking work over a long period of concentrated effort.
- Once they build the product, it can be difficult to change it. In this case, a fundamental alteration to the design of the interface was being suggested.
- None of us likes having our work criticized, especially when we have put in a huge effort and been creative. The builders group is vulnerable to criticism by the other groups as they are responsible for the products.
- When we offer criticism of other's work, we need to consider the three points above. In this case, the tester phrased the criticism very tactlessly. How do I know? Reader, I was that young tester and I look back with embarrassment!

We have seen in Chapter 3 that customers need software to carry out their tasks, and that they want to do this as effectively and efficiently as possible. We saw in Chapter 4 that managers contribute by planning and controlling resources, budget, and time scales; by acting as an information conduit for decisions, to ensure that the proper communication takes place between people in the different groups; and by negotiating between the groups. The builders will realize the customer's requirements in products and services; this means that they need direct communication with the customers, so the managers should encourage that if it is not happening. A large group of people with a variety of roles and skills contribute to the build. Many people think first of the teams and individuals who design and write the code for the software. One IT manager said to me: "Why are you talking about 'builders'? 'Software engineer' is a perfectly good term!" However, many other products need to built along with the code itself, and these can be forgotten or neglected; remember our trainer in Chapter 1? Another trainer's comment is that the request for training material to be designed and built is often made very late in the project:

> I am so tired of being asked to write a training manual in two weeks, with supporting training database, and exercises from scratch, when the software has taken months to write, as though training development is easy to do and takes no time. The software is wonderful; don't they want the training to do it justice?

So, in this book, I emphasize that it is not just the code that is being built, and not just the software engineers who are involved. You might ask, "Will the software engineers write the training material, help guides, and operation support manuals? Those need building too!" Of course, the software

engineers could build all the products, but we need to ask ourselves if they will have time and whether it is reasonable to assume that they have the skills required. For this reason I am using the term "builders" and include in that category all the people who build products, whether these are delivered to customers (software systems, training, user documentation), supporters (operations and support guides, system specifications, regression test packs), or only intended for use during the project itself.

5.2 Who could be in this group?

Builders are the people who specify, design, and build the software and other products. The group includes a number of specialists:

- Those who work on the design and build of software systems; sometimes referred to as the development team (see developers in Chapter 2, Table 2.1). In these roles, we find people who describe themselves as business and system analysts, software architects, designers, software engineers, analyst–programmers, programmers, and developers.
- Those who work on the development of textual and graphical supporting information for the code and software system, whether this is online, on paper, or delivered face to face. In these roles we find people who work as technical writers, authors, training developers, trainers, and graphic designers.

They are stakeholders for quality because they build quality into the product. The products include not just the delivered products (for example, code, user guides, training material) but also the interim products (for example, requirement definitions, designs, and specifications).

As well as the main products—software, training material, and so on—built by the specialists, other people will build products required to aid in the build and support of the products; these are sometimes called infrastructure, support, interim, or secondary products. Almost everyone involved will build something, including plans and reports. In this sense, we are all builders. Examples include:

- Supporters build supporting material, such as infrastructure and environments.
- Measurers build measurement material, such as review checklists, tests, and audit checklists.
- Managers build management material, such as plans and reports.
- Customers build inputs to the project, such as requirements and acceptance criteria.

This means that at some point we will all be builders, and we are all responsible for building quality into our own products. What do we mean

by building quality in? To find out, let us next look at the quality viewpoint and framework of the builders.

5.3 Quality viewpoint

Although builders can be seen as date-driven, the pressure to deliver often comes from their managers and customers. Most builders tend to focus on building products as well as possible. In order to support this, they favor the manufacturing and product-based views of quality.

In Chapter 1, we saw that the manufacturing-based definition focuses on the specification, design, and construction of software products. Quality depends on the extent to which requirements have been implemented in conformance with the original requirements, and our success is measured on our ability to follow a process and deliver products against agreed specifications. Builders will adopt methods, processes, and tools that enable them to provide technically excellent solutions and to meet agreed specifications.

A problem with the manufacturing view is that if we focus on verifying that the system is correct to specification, we may forget to validate that we have the right specification. For the builders, the progression through the software development life cycle (SDLC) may move from product to product, with each product building on a foundation of the previous product in the stream. Once the customer's requirements are understood, the software design is based on those requirements. The builders use the design to develop the code and other material such as training courses and user manuals. If the requirements are wrong, the final products are wrong, even though the designs were followed faithfully. Measurers also favor the manufacturing-based quality viewpoint, as we will see in Chapter 6, so builders and measurers can work together to ensure that verification and validation are applied to products and defects are prevented and removed. We will see in Chapters 9 and 10 that there are ways to overcome this problem, by preparing for the SDLC properly: by choosing our SDLC model carefully, and by anticipating that change will happen throughout the SDLC.

In addition to the manufacturing-based view, builders also favor a product-based definition of quality. As we saw in Chapter 1, this is about working to a well-defined set of software quality attributes that can be measured in an objective and quantitative way. Builders can find customers vague about what they want, particularly in defining the nonfunctional attributes of a software system. This is partly because software attributes are difficult to describe and partly because it can be difficult to know what is possible, particularly with new technology. The supporters share the product-based view of quality; they will be interested in product attributes, especially nonfunctional attributes such as throughput, security, reliability, and maintainability, which affect the operability of the delivered software [1]. They will also be interested in the customer's user-based or fit-for-purpose viewpoint, because the supporters operate the help desk. We will look at the supporters in more detail in Chapter 7. It is, therefore,

worthwhile for the builders and the supporters to engage in dialogue early on and throughout the SDLC. Their product-based view pins down vague "wants" to specific attributes. We will see in Chapter 9 how to derive acceptance criteria that allow objective assessment of the quality of the delivered product, using standards such as ISO 9126. These product-based qualities or attributes are also important to builders because the priority given to particular attributes will affect the design and build choices. For example, if maintainability is important, the code may be structured in a particular way to allow changes to be made easily, but if a different attribute, for example, security or performance, is more important, the code may be structured differently and perhaps be harder to maintain.

Builders also hold a transcendent view of quality, as everyone does, based on their "taken for granted" assumptions; in this case, based on their perceptions and their feelings toward a type of software product. For many builders, their transcendent view focuses on technical excellence or use of particular methods; for example, use of a particular SDLC model (see Chapter 10) or use of a particular technique or tool. A clichéd observation of builders (including my observation of my own work in training development, authoring user guides, and, in the past, writing code) is of a group of people who apply their expertise while obsessively pursuing technical excellence at the expense of a return on investment. They enjoy their own technical innovations and yet can be reluctant to change their processes.

We have seen that the builders favor the manufacturing and product-based views of quality. These viewpoints are shared by many measurers, and this leads them to provide technical solutions of the highest possible excellence; their dedication to quality cannot be overstated. However, their enthusiasm for what they do can blind them to the real quality aspirations of the customers, managers, and supporters. I discuss these groups in Chapters 3, 4, and 7, and show how they hold two other quality viewpoints that are quite different:

- The user-based definition says that quality is fitness for use. It is the user-based definition that encourages us to validate *as well as* to verify the system. The builders need to find out "Does the specification reflect what attributes are needed to enable people do their work efficiently and effectively?" This viewpoint is important to the customers and to the supporters, the two groups most affected by the software postdelivery (see Chapter 12).
- The value-based definition is focused on things that impact on the running of the business as a whole. Software quality should always be determined by means of a decision process based on trade-offs between time, effort, and cost aspects. The builders need to find out "What are the budget constraints for this build and how does that affect what we are able to deliver?" Value for money is important for managers and anyone responsible for budgets. The ROI for the software needs to be predicted before delivery, and scope against the budget managed carefully during the SDLC. It is worth noting here that employers of

software engineering graduates complain that the graduates do not have sufficient business and enterprise skills [2].

Along with acknowledging the user-based and value-based views of quality, the builders need to understand others' transcendent view of quality. The builders must take into account the fact that other people's taken for granted viewpoints may well be different from their own. One important point here is to examine the slant that builders may put on the manufacturing- and product-based quality viewpoints. Let us look at some examples. Here are some real remarks from builders, including a developer, an author, and an analyst:

- "I met the specification so there can't be a problem," said Alice Developer.
- "If you introduce a set process, I'll just have to lower my standards to meet it," said Joe Author.
- "The specification defects arise because the testers won't review our specifications," said Jenny Analyst.

These are three comments that contain a nugget of truth but which are also flawed arguments. Let us look at them more closely.

Alice Developer is correct when she says, "I met the specification so there can't be a problem." The code met the specification. Unfortunately, specifications can be out of date, or just wrong. They are written by people, so they will have mistakes and gaps in them. In the translation through the different activities in the SDLC, the specification may have been worked on by several people as it was refined and detail was added. There may have been an initial misunderstanding that was compounded over the SDLC. Additionally, the customer's requirements may have changed during the SDLC. This happens so frequently that we might almost take it as a given—the requirements will change before delivery.

Corrections and changes will be needed, so builders, including Alice Developer, need to expect that. This makes some people wary of documenting any requirements specification at all: "Why write it down if it is going to change? We don't have time to keep updating a specification!" But it is useful to write down the original specification and changes made to it. Then we have a record of what has changed and why. Additionally, written specifications can be checked for consistency, completeness, and correctness. To encourage keeping specifications up to date, I like them to be as brief as they possibly can, to encapsulate the facts and satisfy any standards or contractual requirements. There is no virtue in writing long documents for their own sake. Some agile methods [3] suggest using media that restrict the length of the documents, for example, A5 cards. Keeping a record is worthwhile. I have been in the situation with technical documentation, for example, where the structure and material have changed back and forth over time. In order to explain to the managers and customers why I was behind time against the plan and over budget, it was useful to have a record of why

changes had been requested as well as how long they had taken. Before I started keeping a written record of what changes were requested, it was easy to get into arguments about why changes had been made and the amount of time it had taken to get from A to B to C and back to A again.

So change is to be expected, but do we want the builders to just make those changes? Alice Developer was working within a contract and a budget, and had done exactly what was asked within those constraints. If she had made the changes requested, she may have exceeded the budget and time constraints, just as I was doing with the technical writing. We both needed to make some improvements in our process:

- We needed to assess the technical impact of the change requested; How big was it? How risky was it? How necessary was it?
- We needed to involve other groups in our assessment: the customers, managers, supporters, and measurers all would be able to contribute to the discussion about whether the change was necessary and what its impact would be (risks, changes to budgets and estimates, effect on other work, what would not get done if this took priority).
- We needed to get agreement on the change.

It is worth remembering that the impact of a change may be much larger than the immediate obvious change. One colleague was changing a training course recently. It looked like a simple change to combine two exercises into one. In fact, combining the two exercises into one document was quite simple, but far more time needed to be spent on changing the references in the slides, contents lists, tutor's notes, printing and binding instructions, and so on. In the same way, a one- or two-line code change can lead to changes in training material, help messages, support and user documentation, installation guides, code comments, and other code modules.

"If you introduce a set process, I'll just have to lower my standards to meet it,"said Joe Author, an excellent practitioner with very high standards of work. Why was the suggestion of introducing what measurers and managers saw as a quality improvement (the use of defined, documented processes) seen as a threat to quality? Many builders see the introduction of standards as either a descent to the lowest common denominator, or as the introduction of bureaucracy that will prevent them from carrying out their work. If processes are poorly designed or badly introduced, this may be the case. What can the builders do to get the processes they need? The best way is to get involved in the work of the designing and implementing processes, so that they own them. The experts who believe that introduction of standards and processes will cause them to adopt poorer methods are the very people who design improved processes and to train others in using them.

Builders need sound processes at all stages to support them in the management of change and the development of low-defect and appropriate products, and this includes use of QC processes on specifications to make them as correct and complete as possible. The cost of changes to fix defects

that could have been found earlier, saving rework, is a useful fact for builders to offer to customers and managers when asking for improvement in the specification of systems. Measurers can help them with this fact gathering. We will see in Chapter 10 that there are SDLC models that provide more opportunities both for checking that specifications are correct and complete, and for adapting to change during the SDLC, and in Chapter 12 we will discuss the difference between fixing defects and making enhancements.

In saying "The specification defects arise because the testers won't review our specifications," Jenny Analyst, the builder, has misunderstood where defects arise in products, and the tester has misunderstood that part of their role is to help the builder. In fact, this was just one comment in a catalogue of woe; the builders and measurers had a complete breakdown of relationship and simply did not cooperate at all. We will return to this story in Chapter 6.

Although measurers and builders can appear to exist on different planets, one reason that they come to loggerheads is that they are so similar. They share a manufacturing-based view of quality, with its emphasis on defect removal. It can happen that the testers' glee in finding defects is matched by the developers' irritation at being caught. One way to improve this is for the testers to couch their defect reports more politely. However, as this is the builder's chapter, let us think about what the builders can do to help. I have seen a number of things that have helped build the relationship between the groups:

- Workshops and training for the builders and the measurers together, where they can discuss what reviews and testing are needed and how these should be done.
- Clear roles and goals for reviews, as we discuss in Chapters 8–12, so that everyone understands what needs to be reviewed and why. The measurers and builders share responsibility for setting the goals and making the plans for the reviews.
- Review of the measurers' plans, test designs, and reports by the builders, thus allowing a reciprocal arrangement.
- Builders welcoming defect reports on their products (sometimes this might be through gritted teeth, as we all know when our own work is reviewed). Each identified defect is a chance to improve something—this product by correcting, future products and processes by introducing defect prevention methods, and the defect reporting process itself.

Communication is key, as always. In a recent software testing course, the group was an equal mix of developers and testers who worked together. I asked the developers and the testers two questions each:

- What information did the developers need when the testers sent them a defect report?
- What information did the testers send with a defect report?

- What information did the developers return with a completed defect report?
- What information did the testers need when a completed defect report was returned?

When we put the results up on the flip chart, there was almost no relationship between the information supplied and that actually required. As soon as the two groups had a chance to reflect on what was happening, they were able to see how to improve the defect resolution cycle. The important information that each group needed was *not* obvious or intuitive to the other group. The developers needed to explain to the testers what information they required in the defect reports. The testers needed to explain to the developers what information they needed when a defect report was returned to them for retest.

In Table 5.1, we see how an example builder, a business analyst, might consider other quality viewpoints. We will look at communication between groups in Section 5.5; here, we are considering what questions about

Table 5.1 Analyst Considering Other Viewpoints for Quality Impacts

Group and Quality View	*Possible Routes in*	*Example Questions*
Customer User-based quality view	Find out the importance of *fit for purpose* on particular features, functions and attributes	What are the goals for the product; what do you want to achieve? What will be the impact on your work if this goes wrong? If it is not available, can you work without it? How much extra time would that take?
		Are there alternative ways you could achieve the same results?
Supporter User- and product-based quality view	Find out the importance for operability and *fit for purpose* of important *attributes*	What will be the impact on your work if this goes wrong?
		If this attribute is not available, can you work without it? How much extra time would that take?
		What is the minimum level of (reliability/security/and so forth) that is acceptable?
		If we delivered X, would that make your life easier or more difficult? What information do you need to be able to support the products? What service-level agreements do we need to consider/support?
Other builders Product and manufacturing quality view	Find out which attributes are technically possible	What information do you need in the specification in order to build the (code/training/support documentation/user guide/installation guide)
		How quickly do you need that?
		Can you build to an early draft expecting changes?
		Which attributes/technical areas will pose particular problems?
Measurers Manufacturing quality view	Find out what will make the product measurable and, hence, testable	What information do you need from me in order to review the document/test the software/audit the process?
		Which functional/nonfunctional areas have given rise to most problems for testing and where have you identified defect hot spots?
Manager Value-based quality view	Identify the *cost* of building and changing the specifications and the *cost* of repair/rework	The cost to the customer and supporter of *not* making the change is X, and the cost of making the change is Y. The cost impact if we make the change and it goes wrong is Z. Can we afford to make the change? Can we afford to not make the change?

quality we might consider in order to understand other people's assumptions about quality as a means of improving understanding.

Particular people in the builder group may have other quality views. In my experience, trainers and technical authors may be closer to the customers in their view of quality than the developers and business analysts. More experienced developers can often be very close in quality view to both the supporters and measurers in their organization.

5.4 Quality framework using the EFQM Excellence Model

5.4.1 The EFQM Excellence Model and the builders

There are a number of well-accepted process frameworks that specifically address the work of many of the roles within the builder group and how those roles interact. Within a particular organization or team, requirements gathering, business analysis, systems analysis, design, and coding work may be carried out by different people, with the work being passed from one person to another; or one person may undertake several or all of the roles. We will look at several generic SDLC models and a tailored model with roles in Chapter 10. In this chapter, I want to set the builders in a framework based on the EFQM Excellence Model [4].

As we saw in Chapter 1, there are a number of frameworks for software work, including the SEI's CMM® family [5], and means of implementing its requirements, such as TSP [6] and PSP [7]. Additionally standards bodies such as IEEE [8] and BSI [9] provide software engineering frameworks and standards. The IT Infrastructure Library (ITIL) has a framework for application management [1], which includes not just application development but also management and optimization of the application in use. This sets application management within an EFQM framework, and also discusses the similar Baldrige framework used in the United States. Other builders also have standards and process frameworks; for example, there are style guides for authors, such as *The Chicago Manual of Style* by the University of Chicago Press [10].

Each organization needs to draw from these models and frameworks to develop processes and standards in a framework appropriate for the type of work, the risks, the experience levels, and the customer expectations of process. For some customers, particular standards, methods or techniques are required, as discussed by Reid in [11]. Some customers, perhaps requiring an engineering approach to control risk, may demand achievement of a particular CMM® level with particular associated processes. Other customers may face risks in achieving time to market; there may be pressure for a fast-track process, which might lead to a different set of processes. The builder's quality framework must match to the customer's quality framework so that the delivered products are not underengineered and unfit for purpose, nor overengineered and too expensive.

The builders are suppliers, and any supplier's ability to recognize and adapt to their customer's quality framework is a critical success factor in the effectiveness of their relationship. The framework for the customer translates into departmental goals and, finally, into the personal objectives and targets for an individual system user, but it also translates into a complementary subset of the supplier's goals, and, hence, into personal objectives and targets for all the builders. This quality framework will drive what the customer needs from the software.

In Chapter 1, we looked at the EFQM Excellence Model and how it is divided into nine parts: five enabling criteria and four criteria for measuring results. In Section 5.4.2, we will look at how the EFQM Excellence Model enablers could be interpreted for builders, and in Section 5.4.3 we will look at builder results. Remember that this model is based on the fundamental concepts of excellence we discussed in Chapter 1 and that equivalent models such as the Baldrige model are available.

5.4.2 EFQM Excellence Model enablers for builders

5.4.2.1 Leadership

We saw in Chapter 2 that the Motivation Survey carried out by Warden and Nicholson [12] showed dissatisfaction between groups. One interesting finding is that although technical experts, for example developers, want to organize their own work, they often feel isolated because they lack feedback from their managers; they lack leadership. In the absence of leadership from their managers, people will either become demotivated, or they will look within their peer group for a leader. I have observed situations in development teams and in authoring teams where the actual line manager is not the natural leader of the team and may even be bypassed. I remember one excellent developer, Dave Tech-Expert, who disliked his manager for being weak both as a person and technically "He doesn't understand my work and he asks stupid questions, but he never listens and he never supports me in meetings," said Dave. Over the years, Dave built a strong relationship with a specific customer team, based on mutual respect and understanding of the technical difficulties in that customer area. The result was that, regardless of agreed-upon project plans, both Dave and the customer bypassed the manager to organize the work that actually happened in that area. The manager continued to report against the project plan, not realizing it was irrelevant; Dave had simply agreed to anything that would make his manager be quiet and go away. A vicious circle was in place, with each action by the manager, the customer, and Dave compounding the mutual suspicion (Figure 5.1).

Warden and Nicholson [12] found that the builders group needed leadership, but that their leaders had to have some technical expertise and understanding, so that the builders knew that they could trust the leader to make intelligent decisions. If they do not trust their managers, they will adopt unofficial leaders, sometimes known as *influence leaders.*

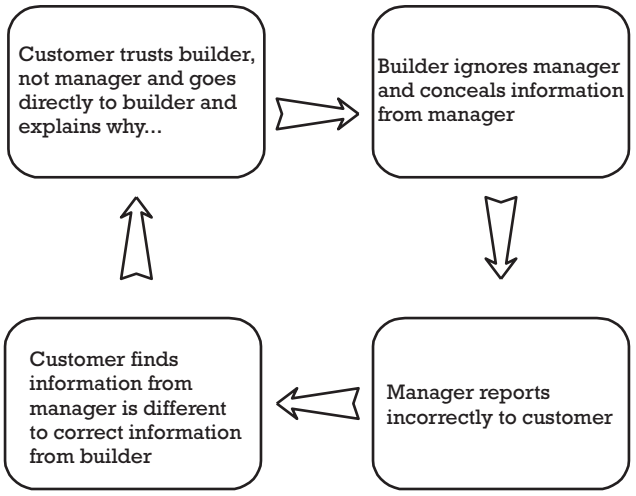


Figure 5.1 Vicious circle of poor and undermined leadership. (*After:* [12].)

5.4.2.2 Policy and Strategy

Builders need clearly defined policies and strategies, which set down what is expected of them and provide direction. We saw in Chapter 4 that managers set policy and strategy based on the higher-level policy and strategy for the organization. The builders need policies and strategies that not only state *what* must be done but also *why* the policy and strategy has been set. We saw in Section 5.4.2.1 that builders will work autonomously and may bypass anything they see as unnecessary or obstructive; generally, we have here a group of intelligent, well-educated, independently minded folk. It is best to set the policy and strategy after consulting with these experts.

Policies will be needed to cover the basic rules for approaching build activities, including rules for tailoring processes. Builders are often asked for fast responses (remember our trainer in Section 5.1?), whether to deal with an emergency, a change in circumstances, or do something that we forgot in our earlier planning. This means that if we have a policy and strategy that describes a normal set of build activities, we should also put together a policy and strategy for deciding when and how to provide a fast-track service, and how to decide how the risks of choosing the fast track weigh against the risks of using the normal approach. Of course, this might be a policy that says "we never tailor the processes," but that is unlikely to be successful. We will see in Chapter 10 an example of tailoring the SDLC. In this case, the policy might state: "A tailored SDLC with reduced steps may be used if the risk of failure has an impact of less than X," and the strategy would show how to tailor the life cycle based on experience, knowledge, clarity of problem definition, speed of response, and risk of failure.

Specialist builders should put together their own framework, with the agreement of the other groups and based on the appropriate practices and published standards, such as those described in the BSI SSQF framework [9] and in CMM® [5].

5.4.2.3 People

We have already noted in Sections 5.4.2.1 and 5.4.2.2 some of the characteristics of builders. This group faces the major challenge of using their creativity and skills to realize the products and services, and their work is measured and judged by the other groups.

The Motivation Survey [12] found that many IT staff want work that has a high motivating potential score, that is, it scores highly for things like the variety of skills required, autonomy of work, significance of the task to the organization, and the chance to complete a task from beginning to end and see the outcome. The survey also found that the range of work went from roles that were excessively understimulating to work that was more complex than it was possible to carry out. The latter was exacerbated by the tendency of IT people to automate parts of their tasks that are repetitive and boring, thus leaving them with only the complex parts. They are actually making their jobs harder.

Builders need a mix of skills, and these would generally be provided within a team rather than by a single individual. Their education, training, and work experiences will of course include all the technical skills and understanding of the methods, techniques, and tools for their tasks. It is worth considering the communication and team analysis techniques, for example, Belbin team roles and the MBTI analysis, discussed in Sections A.1 and A.2, as well as technical skills, when putting together a team. Builders' education, training, and work experiences should also include an understanding of:

- The measurement techniques they will need to apply to their own work and to other people's work. Not only do builders carry out QC activities such as reviews and testing, in which they will need training, but, in my experience, they also make excellent QA reviewers and auditors if trained in the techniques. I noted before that builders respect technical excellence, so having the technical experts carry out process and product reviews across projects spreads knowledge and increases the probability that the QA and QC processes will be accepted as useful.
- The management skills they require for control of their own time and work within a team. This is the focus of the TSP [6] and PSP [7], which emphasize the planning, control, and reporting required by individual software engineers and by teams.
- The interpersonal skills (sometimes called soft skills) to improve their communication with all the groups, including the builders themselves. Kent Beck [3] describes five values without which XP (eXtreme Programming) will not work. I think they are values without which any human endeavor will not work: good communication between people; simplicity in choice of action; concrete feedback to aid communication and simplicity; courage to confront and fix problems, if necessary by

discarding work and starting again; and, finally, what Beck describes as the *deeper value* of respect for others.

- Their customer's goals and strategy in order that they focus their technical, measurement, management, and interpersonal skills toward solving the organization's real problems. This is increasingly important. For example misalignment of IT to the organizations' goals is cited in the McCartney report [13] as a reason for frequent IT project failures. I recently asked a number of IT customers in different organizations what they wanted from IT people. Although some of the responses asked for greater engineering or technical skills, others wanted greater business awareness. One customer said, "Perhaps one danger I perceive in the competency-aligned improvement model (which is generally strong) is that the focus may be 'professional' rather than 'entrepreneurial.' By that I mean more mature business/systems analysis is seen as an end in itself, rather than a means to doing things faster, smarter, and cheaper and making more money for us all."

5.4.2.4 Partnerships and Resources

As we saw in Chapter 1, partnerships cover people outside our team, our project, or our organization with whom we need to exchange information or cooperate with in some way. Resources are the things like IT environments, information, and equipment that we need to carry out our work.

Builders have partnerships with all the other groups and within the builder group, both to exchange information and because some work may be done by others and passed between individuals. They may also have partnerships with builders in other organizations. Some examples of typical partnerships might include:

- Between groups: with the customers and supporters to define the problem to be resolved and share information about possible solutions, risks, and the fit-for-purpose, user-based quality view;
- Between groups: with the measurers to define quality measures for products, carry out QA and QC processes to improve products and processes, and to resolve defects;
- Between groups: with the managers to understand constraints on time, budget, and resources, and the impact these may have on technical risk;
- Within the builders group: developers with designers and analysts to share information and review products concerned with building code;
- Within the builders group: software engineers with training designers and authors to share information and review products concerned with building supporting material such as training courses and user guides;

- Outside the SDLC, with builders and supporters for other projects to negotiate sharing of scarce resources such as environments and tools;
- Outside the organization, with third-party suppliers that the organization has selected to outsource some of the IT work.

5.4.2.5 Processes

As we saw in Section 5.4.1, there are a number of published frameworks for builders' work, and these tend to be biased toward process definition rather than to the other enablers that we looked at in Sections 5.4.2.1 to 5.4.2.4 above. The difficulty that software engineers in particular have is not finding a process definition, but in finding the right process definition for their circumstances. I am not going to describe the candidate processes here; each organization needs to select from those available and will undoubtedly want to adapt the chosen model to their own circumstances. I just want to note that we need to consider a number of factors in choosing a process. These include:

- *Builders' buy-in:* Do the builders understand the proposed process? Do they need training, support, and mentoring? Will the process bring them benefits and do they understand those benefits? If there are parts of the process that make life more difficult for the builders, is there a rational reason for them in terms of benefits to the organization as a whole?
- *Other groups' buy-in:* Do the other groups understand the process? How much are they involved in the process? Do they need training, support, and mentoring? Will the process bring them benefits and do they understand those benefits? If there are parts of the process that make life more difficult for them is there a rational reason for them in terms of benefits to the organization as a whole?
- *Flexibility and appropriateness of the processes:* If the organization has many types and sizes of project, is it clear and easy to tailor the processes? Has paperwork and bureaucracy been minimized? Is there enough documentation and not too much? Is there sufficient control to minimize technical risk and human error, balanced against the process not being excessively expensive to run? Is it easy to change and update the process? Does it meet any external process requirements? Is it easy to identify entry and exit criteria at each step? Are communication points where information is shared identified? Are responsibilities and authorities identified? Does the process allow for changes in the customer and supporter requirements, and for changes caused by rectifying mistakes?

The choice of process needs to balance the needs of different groups—part of the reason for having an agreed-on process is to give assurance to all the stakeholders that they can trust that the right things are being

done. The different quality views of groups means that their expectations of what *should* be laid down in a process will vary. For example, when the builders design their processes, they will need buy-in from all the groups, by showing how the process will support them:

- Customers need the process to help them get what they need to solve their continuously changing problems and aspirations, and to know how the process will meet the demands of the organization, its customers, and society as a whole—all part of "fit-for-purpose," user-based quality.
- Managers need the process to support them in controlling budgets for cost, time, and resources within and across projects, to see how it will help them deal with change and offset technical against financial risks, and for it to support improvements to future planning, estimation, and control—all part of financial/corporate, value-based quality.
- Supporters need the process to include their involvement both in setting requirements and acceptance criteria for product attributes. They need assurance that the process will result in software that can be supported within the IT infrastructure and the IT service and application management standards—all part of "fit-for-purpose," user- and product-based quality.
- Measurers need the process to acknowledge their focus on defect identification, removal and prevention. They need to agree on how measurement processes (QA audits, QC reviews, and testing) are included in the process—for example, that they take place early enough for them to be effective and efficient—and mutual agreement on what information needs to be communicated and when, including information about change.
- Builders need to know that their tasks are included in the process, with links to their own processes early enough for them to complete their products, and with mutual agreement on what information needs to be communicated and when, including information about change.

5.4.3 EFQM Excellence Model results for the builders

5.4.3.1 Customer Results

Customer results are the measure of the effect we have on our customers. Who are the builders' customers? In fact, as well as the customers for the software itself, all the groups are customers of the builders. If we look at what the EFQM Excellence Model [4] refers to as customer perception results and customer performance indicators, we see that perception results are measured externally; we ask people outside our group for their views of us. Customer performance indicators measure what actually happens to our group; whether we get repeat business from a customer,

for example. Both measures are needed so that we see the difference between a customer's reported perception: "We think you are doing a great job," and whether the customer actually *invites the builder back* for future projects. Examples of the type of results that could be measured are in Table 5.2.

5.4.3.2 People Results

In this case, the People Results measure what the builders think about themselves. As with the Customer Results, we measure perception and performance.

We have already seen that this group has quite specific motivational needs [12]. They need a stimulating environment and yet need support and leadership so that they are not alienated.

We will discuss SDLC models in Chapter 10, but I want to note here that some of the models encourage projects with team members involved throughout the SDLC and with frequent feedback loops, whereas others take a more production-line approach. I remember that in one organization, the business analysts would be involved in the first two to six weeks of a project to gather requirements, and after that would move on to their next project. They never saw the outcome of the projects. One person remarked to me, "I don't even know if any code is built as a result of my work, let alone whether it is delivered and worked. I don't know if I solve people's problems." This is the essence of the alienating production line—"People frustrated by work that goes nowhere and an increasingly intrusive system of management that means they have less and less control over what they do" [14]. SDLC models that encourage involvement throughout the SDLC, allow interaction and feedback between the groups, and allow the teams to make decisions and to control their own work, will provide a better working environment for builders. According to Tom Gilb (at a seminar on Evolutionary Delivery in London in September 2003), the frequent-feedback loop is particularly important in improving individual and group motivation. It seems reasonable to suppose that there may be improvements in individual's motivation using models such as XP, which encourage such feedback

Table 5.2 Customer Results—What the Other Groups Think of Builders

	Example of Possible Measures
Perceptions (external to the group), surveys, interviews, compliments, and complaints	Have the builders got a good image?
	Do other groups report that they find the builders easy to deal with (for example, helpful, flexible, honest, and proactive)?
	Are the builders seen as reliable, providing good value, providing a good service?
Performance indicators—measurement of outcomes	Do we get asked back for additional projects?
	Are people proactive in coming to us for help?
	Are our products usually accepted and used over a reasonable life span?
	Do other groups complain about our work?

loops, compared with the waterfall model, which has a production-line approach.

We mentioned in Section 5.4.2.3 that IT staff require stimulating work. One colleague described to me what happened when his teammates got bored; they had a competition to see who could use the most programming constructs in their programs, not because they were needed, but just to try them out. The result was programs that functionally did what they were supposed to do, but did it in a very convoluted way. I am glad I am not maintaining those programs! A measurable result of the boredom of those people was the number of lines of unnecessary code added for fun. Examples of the type of results that could be measured are shown in Table 5.3.

5.4.3.3 Society Results

For the EFQM Excellence Model, society means society at large, and although we might want to measure this, we could interpret "society" for a team as only including their organization. Again, we consider perception and performance indicators. Examples of the type of results that could be measured are shown in Table 5.4.

Table 5.3 People Results—What the Builders Think of Themselves

	Example of Possible Measures
Perceptions—"external" measurement of the group viewpoints by the organization asking for opinion through surveys, interviews, compliments, and complaints	Has a motivational study provided positive feedback? Are builders satisfied with their career paths, rewards?
Performance indicators—"internal" to builders group. Measurement of people's viewpoints by their actual behavior/attributes	Does the builders group match up to the required qualifications and competencies? For example, do they have appropriate BCS qualifications [15] What are absenteeism and sickness levels compared with other groups? Are trainers showing a different absence pattern to developers? What is the level of staff turnover compared with other groups and do people want to be recruited into this group? For example, if we advertise a business analysis course internally, is it over- or undersubscribed?

Table 5.4 Society Results—How the Builders Relate to Society

	Example of Possible Measures
Perceptions—external to the group—surveys, interviews, compliments, and complaints	Are the builders seen by the organization and wider society as acting ethically? Do the auditors and testers check that the systems act properly with regard to health risks, safety, hazards, and the environment?
Performance indicators—internal to builders group	Has the group won any accolades, for example, for paper- and energy-saving initiatives, in line with corporate initiatives? Has the group been recognized for its efforts to reduce security risks internally and externally, including customer data confidentiality, prevention of spam and virus attacks, and IT governance provisions?

5.4.3.4 Key Performance Results

The EFQM Excellence Model divides key performance measures into two areas: financial and nonfinancial measures. Some examples are shown in Table 5.5. It is important to reflect on *why* the organization needs an IT group; unless software is itself the revenue-earning product of the organization, the only reason for the software is to service the needs of the organization, allowing its customers and users to carry out their work more easily. For a business, this will mean that the key performance results for IT and, hence, the builders, must reflect the commercial drivers for the organization. For a nonprofit organization, again, the builders must support the organization's goals, increasing the efficiency and effectiveness of the services and products provided.

5.5 Communication between the builders and other groups

We will see in Chapters 8 to 12 that builders are involved in the whole life span of a piece of software, from its conception, through the software development life cycle (SDLC), during delivery, and postdelivery until decommissioning.

Builders have a large responsibility in communication to and from the other groups. They need to understand what is required, and they need to communicate back what is possible and what has been achieved. A problem with this is that IT people may have quite different personality types and communication styles than their customers [16–18]. It appears that a larger proportion of IT people tend to be introverts, working by intuition, thinking, and judgment skills. If across the general population and, therefore, among the builder's customers there is a bias toward extroverted behavior and working more by perception or emotional ways of thinking, we can see that there will be a clash of communication styles. I have seen this particularly where sales, marketing, or customer service staff need to work with IT. If Phil Marketing-Mann is emoting and waving his arms around, Dave Tech-Expert may just ignore him and, as he has brought his laptop into the meeting, just get on with some other work. One comment in [18] is that whereas many IT people prefer written communication, their customers may prefer

Table 5.5 Key Performance Results—Financial and Corporate

	Example of Possible Measures
Financial	What is the return on investment (ROI) for the software we have delivered this year (cost of software/money saved/number of new customers/market share gained)?
Nonfinancial	Number of projects/customers who request our team to carry out IT-related work, and the size of these projects
	Cycle times for responding to customer requests
	Innovation by the builders (e.g., new technology to support time to market with new products)

face-to-face communication. This alone can give rise to misunderstanding and resentment. This is an important factor in the success of the builders:

> … [P]otential weaknesses for the organization lie in the lack of feeling and perception. These can become key issues if you need more flexibility and a greater focus on the customer for future success. A culture biased toward thinking and judgment is in danger of neglecting how the customer feels about the service he or she is getting, which can be an important factor in the decision to buy. [18]

Of course, in some builder–customer relationships the preferences may be reversed. What is important is that the communication preferences in a particular situation are understood. There is more about personality types and communication in Chapters 8 to 12 and Appendix A. In Figure 5.2, I have shown the communications between builders and the other groups. Builders have information that others need (Table 5.6) and they also need information from other people in order to understand the constraints on delivering their wish list. Table 5.7 lists the information that builders need from the other groups.

Remember, it is not just the content but also the manner of the communication that is important. All the groups need to understand that they will favor different communication styles, and that individuals within the groups may or may not fall into the usual pattern for the group; we must beware of pigeonholing people. The lesson is that each builder needs to understand their own favored communication style and to consider the favored communication styles of other people within their own and other groups. If you are interested in discovering your own preferred style, you may wish to look at the Myers-Briggs Type Indicator (MBTI) and take an MBTI quick test (see [17]).

5.6 Summary of the group

Builders are essential to the quality and success of software and systems, and their technical skills are needed throughout the life of software. However, builders must communicate well with each other and the other groups if they are to be effective. This means considering how they demonstrate

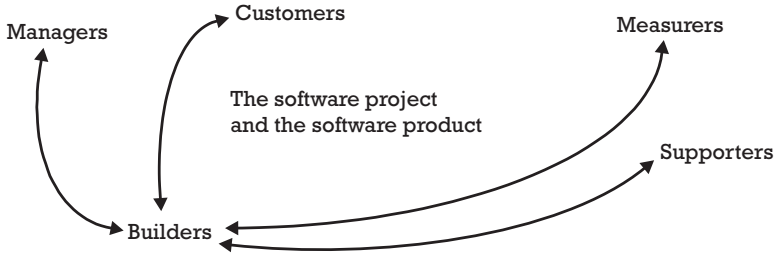


Figure 5.2 Communication between groups.

Table 5.6 Information That Builders Have That Others Need

Before the SDLC starts and updated throughout the SDLC	Technical constraints
	Technical innovations
	Technical risks
During the SDLC	Results of QC done by the builders themselves
	Technical risks and areas on which to focus QA and QC work
	List of known problems when products are delivered to QC or go live
	Progress, problems, changes to risks, changes to constraints
At delivery	List of known problems
	List of incomplete products
	Delivery and installation requirements
Postdelivery	Evaluation comments
	Improvement ideas

Table 5.7 Information That Builders Need from Others

Customers	Business constraints
	Business risks
	Business requirements
	Evaluation of quality of system when it is in use
Managers	Financial, time, and resource constraints
	Progress toward goal
	Changes to goals and constraints
Other builders	Progress with products (different builders will be responsible for different products: specifications, designs, code, training material, and so on)
	Changes to products
Supporters	Technical constraints for live system
	Acceptance criteria
	Evaluation of quality of system when it is in use
Measurers	Acceptance criteria passed
	Defects and nonconformances identified

that they provide results and assurance for other groups, understanding other's quality views and communication styles, and explaining their own quality view and preferred communication style in a way that is understandable to the rest of the team. People with other communication styles and preferences, within and outside the builders group, should also have consideration for the builders' preferences as individuals. This will help to ensure that that the good communication that is essential for the builders' success is maintained.

References

[1] IT Infrastructure Library, *Best Practice for Application Management,* London, England: Office of Government Commerce, 2002.

[2] Holcombe, M., and M. Gheorgha, "Enterprise Skills in the Curriculum," *Ingenia,* Vol. 15, February/March 2003, pp. 56–61.

[3] Beck, K., *Extreme Programming Explained,* Reading, MA: Addison-Wesley, 2001.

[4] European Foundation for Quality Management, "EFQM Excellence Model," http:// www.efqm.org, August 2003.

[5] Software Engineering Institute, "Capability Maturity Model®," http://www.sei.cmu.edu, July 2003.

[6] Humphrey, W., *Introduction to the Team Software Process,* Reading, MA: SEI, 2000.

[7] Humphrey, W., *Introduction to the Personal Software Process*, Reading, MA: SEI, 1997.

[8] IEEE standards; see Web site http://standards.ieee.org/.

[9] British Standards Institute, *PD0026:2003, Software and Systems Quality Framework—A Guide to the Use of ISO/IEC and Other Standards for Understanding Quality in Software and Systems*, London, England: British Standards Institute, May 2003.

[10] The University of Chicago Press, *The Chicago Manual of Style*, 15th ed., Chicago, IL: The University of Chicago Press, 2003.

[11] Reid, S. C, "Software Testing Standards—Do They Know What They Are Talking About?" http://www.testingstandards.co.uk/publications.htm, August 2003.

[12] Warden, R., and I. Nicholson, *The MIP Report, Volume 2: 1996 Motivational Survey of IT Staff,* 2nd ed., Bredon, England: Software Futures Ltd., 1996.

[13] Cabinet Office, *Successful IT: Modernising Government in Action*, London, England: HMSO, 2000.

[14] Smith, P., "Alien Resurrection," http://www.chartist.org.uk/articles/econsoc/jul03smith.htm, October 2003.

[15] British Computer Society Qualifications, http://www1.bcs.org.uk/link.asp?sectionID=574, September 2003.

[16] Kroeger, O., J. M. Thuesen, and H. Rutledge, *Type Talk at Work: How the 16 Personality Types Determine Your Success on the Job,* New York: Bantam Doubleday Dell, 2002.

[17] Team Technology Web site, "Working Out Your Myers Briggs Type," http://www.teamtechnology.co.uk/tt/t-articl/mb-simpl.htm, October 2003.

[18] Team Technology Web site, "The Mother of Strategic Systems Issues: Personality," http://www.teamtechnology.co.uk/tt/t-articl/news1.htm, October 2003.

Selected bibliography

Adair, J., *Effective Teambuilding*, London, England: Pan Books, 2003.

Adair, J., *Effective Time Management*, London, England: Pan Books, 2003.

Brooks, F. P., *The Mythical Man Month*, Reading, MA: Addison-Wesley, 1995.

Caputo, K., *CMM® Implementation Guide: Choreographing Software Process Improvement*, Reading, MA: Addison-Wesley, 1998.

Clarkson, M., *Developing IT Staff*, London, England: Springer Practitioner Series, 2001.

Crosby, P. B., *Quality Is Free*, New York: McGraw-Hill, 1979.

Detiénne, F., *Software Design—Cognitive Aspects*, New York: Springer-Verlag, 2001.

Fowler, M., and K. Scott, *UML Distilled*, Reading, MA: Addison-Wesley, 1997.

Gershuny, J., *After Industrial Society*, New York: Macmillan Press, 1978.

Kruchten, P., *The Rational Unified Process*, Reading, MA: Addison-Wesley, 1999.

Nance, R. E., and J. D. Arthur, *Managing Software Quality*, New York: Springer-Verlag, 2002.

Parker, S. R., et al., *The Sociology of Industry*, London, England: George, Allen and Unwin, 1978.

Schein, E., *Organizational Culture and Leadership*, San Francisco, CA: Jossey-Bass, 1997.

Smith, J., *How to Be a Better Time Manager*, London, England: Kogan Page, 1997.

CHAPTER

6

Roles and Quality: Measurers

Contents

In this chapter I shall:

- Introduce the members of the measurers group, their roles, and activities;
- Introduce their quality viewpoint;
- Provide a framework for the measurers' activities within the EFQM Excellence Model;
- Identify information flows between the measurers and the other groups.

I'm going to buy a magic wand, and then when the development manager says to me, "We've finished the build, now can you do the quality stuff," I can just wave the wand and make it happen.

—Quality assurance manager complaining about the way quality activities are regarded in projects

6.1 Introducing the measurers

6.1.1 Why do we need QA and QC?

Software projects generally include some activities such as testing, to check the software before it is released for live use, and auditing, to check that appropriate processes are followed during the project. We need to do this because humans make mistakes. This includes making poor decisions and building things with faults in them. Once we have made a mistake, it is often difficult for us to find it, so we ask someone else to check what we have done. A software project is generally quite complex, with many decisions to be made and many products leading to the final delivered software. This means that there are lots of opportunities for people to get things wrong. They may

misunderstand each other, or make wrong assumptions, or simply slip up in what they are doing.

During the software-development life cycle (SDLC), testing and other quality activities are used to identify and prevent mistakes. So why was our quality assurance manager frustrated? The problem was that the development manager believed that the QA and QC activities introduced quality into the products and, therefore, that the QA manager was responsible for the quality. As we saw in Chapter 1, quality assurance activities (QA) are used for checking processes, and quality control (QC) for checking products. The output from QA and QC activities is a set of measures of quality. In fact, QA and QC activities do *not* change the quality of anything; all they do is to provide information about the quality of the processes and of deliverables from those processes, in order that the processes and products can be changed.

6.1.2 Just measurers or also improvers of quality?

Someone recently commented to me that "The test group does improve the quality by finding faults (early) in the requirements and later in the design and code. I do not think we spend up to half of the development budget just to measure quality," so let us consider that point.

I would say that the test group has *contributed* to the quality of the delivered product by identifying faults, and the earlier they find the faults, the greater the value of their contribution, because of the money and time saved. But the testers have not *changed* the quality of the deliverable; the deliverable does not change when the tests are run or the results analyzed.

The tests have provided information about the quality, that is, the identification of a number of faults and the areas of the product that meet their acceptance criteria. If the products are changed or if a suitable workaround to the defect can be provided, the tests have contributed to reducing failures during use.

This is true also for other forms of QC, for example, document review. The process of the review itself gathers information about the quality of the product under review, but does not in itself change it. However, the document reviewers contribute to the quality of the document by suggesting improvements and by identifying defects.

As a result of the testers' or reviewers' measurements, the team as a whole (customers, managers, builders, supporters and measurers) can make a decision about how to deal with the faults; they may be removed, or worked around, or ignored. That decision will be driven by the user-based quality view (is it fit for purpose?), the value-based quality view (will the cost of fixing it be less than the cost of leaving it?), and the transcendent quality view (do we like it?), as well as the manufacturing-based quality view (are there defects in the product?), and product-based quality view (does the product have the agreed attributes?).

We spend a large part of the budget on testing and other QC activities to understand all these quality views; we need information about the number of defects in the product to give us a view of its manufacturing- and

product-based quality, so that we can form what will be a decision based on all the quality views.

Similarly, those measurers involved in looking at processes, for example, auditors, carry out QA tasks that provide information about processes, and a decision is then made about whether to enforce or change those processes. QA activities measure both the appropriateness of the chosen process and how well the process is adhered to. An audit or process review does not, in itself, change the processes; it simply comments on them. Auditors will also measure how closely a process adheres to standards and to legislative or regulatory demands. We hope that by choosing processes wisely and adhering to them, we will prevent people from making mistakes, and, hence, prevent them from breaking mandatory requirements or building defects into products.

Preventing defects in the products reduces the places where failure can occur when a product is used, and the process improvement aspect of QA is an important one. However, the people in this group, *while in this group,* can only observe, measure and recommend improvements in the process. It is the people who actually carry out the work who implement the process improvements, thus preventing defects.

This is why, instead of calling this group the testers, or the QA group, or the QC group, I have chosen to describe them as the measurers. This is to emphasize that their contribution is to provide information for all the groups by measuring the quality of the software, other products, the processes. The measurers support decision making and quality improvement.

These measures may be negative; for example, the number of defects in a product, or the number of times a process was not followed correctly. Additionally, we might want positive measures; for example, the number of requirements successfully delivered or the number of acceptance criteria passed. In Sections 6.1.3 and 6.1.4, I will enlarge on two aspects of QA and QC that do seem to directly affect improvement.

6.1.3 Defect prevention

QC activities need to be carefully timed. We will see in Chapters 9 to 12 how QA and QC activities are used appropriately at each stage in the life of a software system. Before we even start the software-development life cycle (SDLC) we need to decide whether we need an SDLC at all. Our biggest mistake can be to do the wrong project or to build something we just did not need.

We will see in Chapter 9 all the activities we need to do before we start an SDLC and that measurers, whether QA or QC specialists, are vital to this part of the process. Carrying out QA and QC on the processes and deliverables at this stage, for example, on the contract for the project, helps us start the right SDLC, which prevents us from making mistakes later on. The earlier QA and QC are carried out, the earlier defects are removed. These early defects tend to propagate through later deliverables; in other words, a mistake in the outline of the project aims and objectives, or an omission in the delivery list on

the outline plan could lead to many mistakes in the project itself (Figure 6.1). In this figure, I illustrate two paths to problems; in path one, we simply do the wrong project; this one massive mistake leads to all the project activities and products being defective—a costly misunderstanding. In path two, although we are working on the right project, we make some mistakes in defining the requirements, which escalate through the SDLC.

For example, we need a periodic report from the system—(the period may vary)—but it is documented as monthly because that is given as an example during a meeting. This affects a number of design choices. The report is designed to deliver the previous month's data at the month end. The system archive and clear-down is designed to match the reporting cycle,

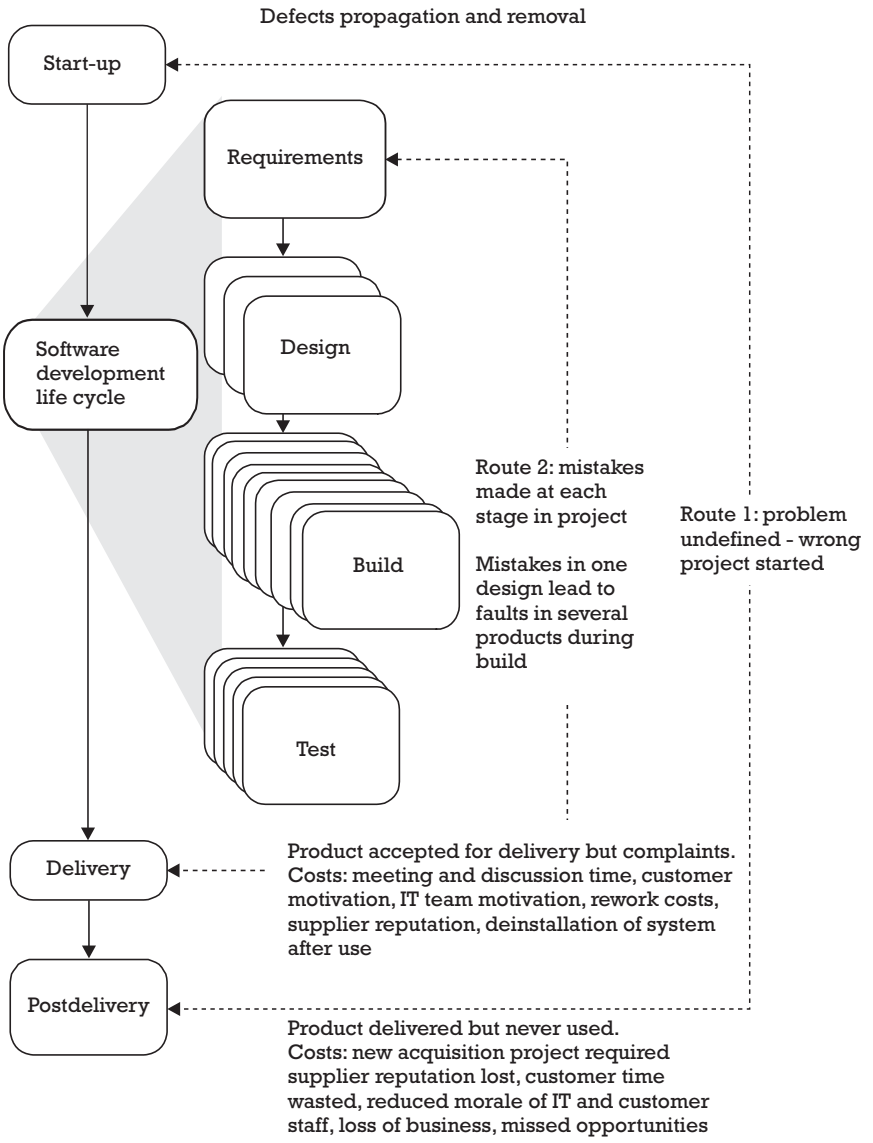


Figure 6.1 Defects propagation and removal.

so that the previous month's data is archived following the report; only 31 table lines are allowed on the current-month data collection fields. The designs affect nine programs, so each of them is wrong. Early tests are based on the requirements and design specifications, so they report no problems, but the user-acceptance test raises defects in all the programs. The IT team do not agree that they are defects, because the software meets specification. The software is accepted for delivery after much argument, but there are complaints after the first month that the reports can only be done at month end, and that data is autoarchived following the report so it cannot be used again. This is fixed, but when one user tries to report after three months, which is their preferred cycle, the data table cannot hold all the data because it is only 31 lines long. The resulting costs include meeting and discussion time, customer motivation, IT team motivation, rework costs, supplier reputation, maintenance to the system, and a recurrence of problems once the "fix" is in place.

Errors made in defining the requirements for the software solution can be found early on in the SDLC, by checking the requirements for defects and omissions immediately after the requirements have been gathered. Removing defects in the requirements prevents them from being propagated into later activities and deliverables such as design and build. We will see in Chapter 10 how different SDLC models encourage QA and QC to be carried out at different times, with a greater or lesser emphasis on defect prevention and defect removal.

6.1.4 The Hawthorne effect

One problem with understanding whether our quality measurements cause improvement or not is the Hawthorne effect. Named after the Hawthorne works of the Western Electric Company in Chicago, where the phenomenon was first observed, it indicates that people change behavior if they know they are being observed, or if they know that a process change is supposed to improve something. One definition of the Hawthorne effect is:

> An experimental effect in the direction expected but not for the reason expected; that is, a significant positive effect that turns out to have no causal basis in the theoretical motivation for the intervention, but is apparently due to the effect on the participants of knowing themselves to be studied in connection with the outcomes measured. [1]

This means that the very act of carrying out QA or QC may change the quality of the product or adherence to the process because people know they are being observed. This change can be positive or negative, in my experience, depending on the maturity of the organization and individuals. Some people will take a view that "someone else is going to check this…" and let their standards slip, whereas others' pride in their work means that they give it extra checks themselves before submitting it to QC or QA.

6.2 Who could be in this group?

Measurers include specialists who spend all their time in this group, and members of the other groups who take on a measurement role for a particular project, or carry out measurement tasks from time to time. Specialists in the measurers group include:

- People who audit or check the conformance to and suitability of processes; their measures are made against process standards—quality assurance specialists (see quality practitioners in Chapter 2, Table 2.1).
- People who measure the quality of the products, including software, using processes and activities to check for completeness, correctness, suitability, and adherence to specification—quality control specialists (see testers in Chapter 2, Table 2.1).

Although many organizations have specialists or teams focused on QA and QC activities, for example, audit and compliance teams, it is unusual for the measurement of quality to be left entirely to specialists. One common reason for this is that there are not enough specialists in the organization to carry out the work. We will also see in this chapter, and in Chapters 8 to 12, which describe the life span of software, that different types of checks are needed at different times. This means that, as required, the customers, managers, builders and supporters may all carry out measurement activities. These might include part-time or temporary membership in the quality assurance (QA) teams and quality control (QC) teams:

- The QA teams carry out activities that check that teams have selected suitable processes and are adhering to them. QA activities might include process review and quality or process audit.
- The QC activities include software testing, software inspection, and product review, and there may be, for example, test teams and inspection teams.

We all check our own work, and so we all measure quality; but specialist measurers hold a particular viewpoint of quality. Let us examine that next.

6.3 Quality viewpoint

Both QA and QC tend to focus on discovering defects against specifications or standards. This means that specialists for these activities favor the manufacturing view of quality. As we discussed in Chapter 1, manufacturing-based quality focuses on the manufacture of software products, that is, their specification, design, and construction. For specialist QA and QC people, therefore, quality depends on the extent to which requirements have been implemented in conformance with the original requirements.

When measurement focuses on faults and failures in products, success is measured by our ability to follow a process and deliver products against

agreed-on specifications, so the report that "the software was built to specification and there are a low number of defects" might lead to a recommendation to release the system for live use, even if it does not do what the customer needs.

We can see this reflected in QA standards such as ISO 9000:1994 [2], and in QC standards, for example, BS 7925-1 [3] and BS 7925-2 [4]:

- An audit against ISO 9000:1994 will look for a documented process and evidence that the documented process is followed. It is concerned that a contract has been agreed on and met, but not whether that contract describes a suitable product for the customer.
- In BS 7925-1, testing is defined as the "process of exercising software to verify that it satisfies specified requirements and to detect errors." Using this definition, we *will* verify ("is the system correct to specification?"), but if we do not take account of the *user-based* definition of quality (see Chapter 3, Customers), we *may forget* to validate ("is this the right specification?").

As well as a manufacturing view, many measurers hold a product-based view of quality. Here, quality is based on a well-defined set of software quality attributes that must be measured in an objective and quantitative way. This quality viewpoint is particularly appealing to measurers because it provides a way of changing concepts which are often ill-defined and amorphous into neat, measurable acceptance criteria. We can use acceptance criteria to objectively assess the quality of the delivered product. Rather than reporting in vague, unquantified terms that "the software appears to be reliable," we can measure and report clearly, "The software is 98% reliable when running continuously over a 7-day period. Recovery time is less than 1 minute at each failure." We can use standards such as ISO 9126 [5] to help us define attributes measurably. These include reliability, usability, security, and functionality attributes. We will see how to do this in Chapter 9.

Most people, and that includes measurers, hold a transcendent view of quality. As we saw in Chapter 1, this means that we "know quality when we see it"; our knowledge is based on our experiences, taste, affections, loyalties, and emotions. Unfortunately, this means different people will have different reactions to a product, so it is hard to agree on what is "right." The transcendent definition for a particular person will be based on that individual's "taken-for-granted" assumptions. For many specialist measurers, their transcendent quality viewpoint is strongly attached to a taken for granted assumption that quality is related only to specification, number of defects, and product attributes. For these people, unless products and services can be measured against an agreed-on standard and meet or exceed that standard, they are inherently lacking in quality. For other specialist measurers, their taken-for-granted assumption is that a pursuit of defects (identification, removal, or prevention) and of product attributes (number and level of implementations) almost regardless of cost or time, must be a good thing.

A clichéd observation of specialist measurers (and I am one myself) is that common characteristics of people in the group are pedantry, obsessive attention to detail, a strongly developed ability to complain, and a delight in other people's mistakes. As a group, we can be difficult for others to deal with and perceived as unhelpful and focused on our own concerns (see Chapter 2).

So the measurers favor manufacturing and product views of quality and within those tend to focus on defects and non conformance. At its most extreme, the quality specialists' manufacturing-based quality view can be rigid and not focused on the overall needs of the customers and the organization. To improve their relationship with the other groups, they must consider the groups' quality viewpoints. As we saw in Chapters 1 and 2, two other definitions of quality reflect the views of the people using the software and those who pay for it. These perspectives are about supporting the needs of the organization and its stakeholders, within the organization's constraints; therefore, what constitutes "quality" may change over time.

- The user-based definition says that quality is fitness for use. It is the user-based definition that encourages us to validate *as well as* to verify the system. The measurers need to find out, "Can people do their work efficiently and effectively using this software?" This viewpoint is important to the customers and to the supporters, the two groups most affected by the software postdelivery (see Chapter 12).
- The value-based definition is focused on things that impact on the running of the business as a whole. Software quality should always be determined by means of a decision process based on trade-offs between time, effort, and cost aspects. The measurers need to find out, "If we release the software now, how much extra will we spend on support in the first month? If we are a month late, what will it cost the organization in fines and lost business? Should we release or do more testing?" Value for money is important for managers and anyone responsible for budgets. The ROI for the software needs to be predicted before delivery and measured postdelivery (see Chapter 12) in order that the organization as a whole can evaluate whether the project was worthwhile.

Along with acknowledging the user-based and value-based views of quality, the measurers must acknowledge that the transcendent view of quality depends on the individual. The measurers must take into account the fact that other people's taken-for-granted viewpoints may well be different from their own view. One important point here is to examine the different slant that builders and measurers may put on the manufacturing- and product-based quality viewpoints. As we saw in Chapter 5, builders, particularly those working in software development rather than other areas of product build, focus on technical excellence as a means for enhancing their work. They tend to be interested in the process aspects of manufacturing and in the enhancement of product attributes. They are also on the

receiving end of criticism from the measurers about their processes and their products, which can cause communication problems (Chapter 2).

Let us look at some examples, looking at one subgroup within the measurers, specialist system testers. These people test software systems, often as part of an independent group within the project but separate from the builders. Here are three real remarks I have heard system testers make:

- Sam Tester: "We will not test without a written specification."
- Jo Senior-Tester: "If we're not aiming for zero defects how many do we want?"
- Jim Test-Expert: "We cannot sacrifice quality to cost."

These views are all correct, yet simultaneously wrong. Let us examine them one at a time.

"We will not test without a written specification." Sam Tester needs the specification in order to draw up tests and expected results for those tests. Without a written specification, it is difficult not just for the testers to proceed, but also for all the groups to remember what has been agreed on. However, simply to refuse to continue until a written specification is provided is unhelpful to the other groups if they had not expected to provide a written specification. The pragmatic approach *for this occasion* might be for the tester to agree to research the expected results by interviewing the customers. The tester then, as part of the measurement process, measures the cost of late development of the specification, in terms of additional work done and the number of defects in the build that could have been prevented had there been a written specification. This can be used to persuade all the groups to agree on improvements in process for the next project. For improvements to be accepted, the tester must provide the information showing how the other groups' quality viewpoints are met. This means that measurements must be changed into information that addresses those viewpoints, to make it easy for others to buy into the suggestion for change. For example, does the software do what the customers and supporters require, or does it stray from a "fit-for-purpose" model? Is it missing things that are needed? From the managers' viewpoint, what was the cost of exceeding "fit for purpose" by adding unnecessary things, and what was the cost and impact of defects found late that could have been identified in a written specification? When talking to the builders, focus on their transcendent view of quality as technical excellence (see Chapter 5), and the aspects of the manufacturing and product-based viewpoints that would be enhanced by a written specification. Would a clearer definition of what is required allow them to concentrate on building technically excellent solutions that will be more easily accepted, thus reducing boring rework? In Chapter 9, we will look at what we can do before we start to build software, to provide a clearer definition of what is required. In Chapter 12 we will look at evaluation of delivered systems.

"If we're not aiming for zero defects how many do we want?" Jo Senior-Tester is concerned that if the testing has not uncovered all the defects in the

products, and that these defects have been corrected, the delivery will be of an unacceptable quality. Also, testers *know in their hearts* (transcendent quality) that they want to deliver a product that is *perfect*. Their argument is, "If we are not looking for perfection, how flawed is OK? With products and services outside software, we don't expect defects, do we?" In fact, customers and supporters are not bothered by defects as such. What they want to avoid is *failures*. By this I mean that the product not behaving as they expect it should. They will also differentiate between high and low impact failures. They need the products to be fit for purpose rather than perfect, and the software defects may not lead to high impact failures. The tester needs to work with the customer and supporter to define acceptance criteria that encapsulate a "fit-for-purpose" view. We will see how to do this in Chapter 9.

"We cannot sacrifice quality to cost," explained Jim Test-Expert. This comment is a continuation from the last one. The tester is concerned that the other groups are *only* interested in the cost, at present, but that later there will be a backlash, possibly directed at the tester, if the delivered products are disappointing. From a manager's viewpoint, cost and quality are inextricably linked in the value-based view of quality. For the customers and supporters, too, the budget they are prepared to spend must be balanced against their wish list. Builders, conversely, can be fixated on delivering additional attributes and see that as adding value and quality. Testing these additional attributes may be a problem for the tester if it adds unplanned scope and cost. If cost is the limiting factor in this project, the tester needs to explain what testing is possible within the budget, what testing cannot be done, and the risks associated with not doing that testing. In some projects, the goal is to deliver the minimum that will allow the customers to proceed—quality in that case is closely tied to fit for purpose and value for money. In other projects, the goal may be reducing risk in a safety-critical situation—quality in that case may be tied to product attributes such as reliability and to manufacturing goals such as defect prevention and removal. If, as testers, we believe quality will be impacted, we need to examine "what aspect of quality is now adversely affected and could be improved by increased spending?" In doing this, we need to not just focus on process, defects, and attributes (manufacturing and product views) but also to provide value, user, and transcendent examples of quality loss (see Table 6.1). The precise questions to use depend on the relationships between the groups, so these are just examples. Using questions allows each group to arrive at its own conclusion, based on its own quality view. As above, defining "fit-for-purpose" acceptance criteria will help prevent this problem, and in Chapter 9 we will also see how to arrive at a common understanding of risks and constraints across the groups.

When considering the quality costs and the impact of QA and QC activities on the project, it is important to remember that quality activities *can in themselves damage quality*. This idea surprises and offends some quality practitioners but it must be faced. Traditional quality economics supports the testers I described above because it assumes that all quality activities are inherently a "good thing." Yes, there is a trade-off between the cost of the quality activities and the cost of the failures that they may prevent;

Table 6.1 Tester Considering Other Viewpoints for Quality Impacts

Group and Quality View	*Possible Routes into a Discussion with This Group*	*Example Questions*
Customer User-based quality view	Find out the impact on *fit for purpose* if things go wrong	What is the impact on your work if this goes wrong? If this feature is not available, can you work without it? How much extra time would that take? Where would you like testing to concentrate? From what you have said, it looks like if we find problems they will have X impact. Do you agree?
Supporter User and product-based quality view	Find out the impact on *fit for purpose* and important *attributes* if things go wrong.	What is the impact on your work if this goes wrong? If this attribute is not available, can you work without it? How much extra time would that take? Where would you like testing to concentrate? What is the minimum level of (reliability/security/…) that is acceptable? What is the postdelivery support budget? From what you have said, it looks like if we find problems they will have Y impact. Do you agree?
Builder Product- and manufacturing-based quality view	Find out the *likelihood* of different areas going wrong	Which attributes and areas would you like testing to concentrate on? Which areas were hardest/easiest to build? From what you have said, it looks like I might find it useful to look at areas A, B, and C. Do you agree? The software has just done this. Is that what you would expect? How difficult would it be to make changes in this area?
Manager Value-based quality view	Identify the *cost* of things going wrong and the *cost* of repair/support.	What budget do we have for support after going live? The impact of not testing these areas is X additional time (hence cost) for the customers and Y additional time (hence cost) for the supporters. The likelihood of them going wrong is Z, so the risk of them going wrong (likelihood times impact) is [Z * (proportion of X + proportion of Y)].

however, the possibility that quality activities can detract from a project is not considered. This view is now being challenged [6] and the distinction is sometimes made between quality costs (activities that contribute to the fitness for purpose) and quality losses (activities that cost money but make no contribution to the fitness for purpose of the product). Adopting this mindset goes some way to wedding the value-based quality definition to those quality viewpoints normally held by measurers. In one organization, a team of measurers were introduced to the concept of "quality loss," and on examining their activities found several examples of value-destroying quality activity. All of the following are failures in efficiency and effectiveness that colleagues and I have seen in more than one organization:

- Inspections and other document reviews for which participants have not prepared. These take time but have a low defect-detection rate. Just "having the inspection" in order to get the sign-off against the quality policy does not improve the quality of the product.

- Project audits carried out by staff who are not politically empowered. These require significant time investments from project teams as well as the auditors, and consistently fail to escalate significant nonconformances from standards. They may even make the situation worse because the political status of the auditors may be used as a reason to prevent measurements being taken.
- User-acceptance testing directly replicating the tests carried out in system testing (at cost), while not identifying additional defects, instead of carrying outtests at different levels, each having their own purpose and focus.
- Practitioners may be trained in quality methods that they consider to be only theoretically valid, and thus they may not implement them on their projects. In one organization, I found that the senior management had sent teams on a test techniques training course as a "treat." The view was that if they had some (any) training, this would keep them quiet for a while and improve their motivation, but there was no intention to allow any changes in working practice as a result of the training. As a result, the teams were more demotivated following their return to their desks because they were not allowed to apply the techniques they had learned. Additionally, a large proportion of the year's training budget had been used.

Carrying out this type of exercise helps eliminate costly but ineffective activities and it helps the measurers evaluate their own activities through the value-based quality definition.

It is important to note that, because this group is so widely supplemented by recruits from all the other groups, many individuals hold wider quality views. The important thing for each individual is to consider which quality view they are favoring, and then consider the other views:

- Some people in this group work mainly in *user-acceptance testing* and will support the user-based view of quality. They may be closer to the customer view and need to take extra time to consider value-, product-, and manufacturing-based views.
- An increasing number of measurers are *test automation specialists*; they can be closer to builders in outlook. They need to make sure that they do not forget the user- and value-based views.
- *Quality managers and test managers* may take on aspects of the manager's group, and be more focused on ROI and value for money, so they should make an additional effort to remember the user-, manufacturing-, and product-based views.
- The work of *audit and compliance groups* might take on aspects of the customers and/or managers groups and consider fit for purpose and ROI as well as compliance to standards and regulations, broadening their view from manufacturing defects to user and value views.

- If the *testers* look after the customer help desk as well as testing products, which does happen in some package suppliers, they become more focused on supporter activities during installation and postdelivery, and will be trying to consider user- and product-based quality as well as manufacturing-based quality.

6.4 Quality framework using the EFQM Excellence Model

6.4.1 The EFQM Excellence Model and the measurers

The measurers are, almost by definition, very interested in frameworks that include defined processes, standards, and specifications for work, including their own. Their measurements take place against that framework of processes, standards, and specifications. Quality assurance standards, including the ISO 9000 series, can be used to define the QA and QC processes as well as the organization's overall processes. There are also specific professional bodies, standards, frameworks, and best-practice guides for all aspects of QA and QC work. Some sources for these are listed in Appendix A, but you should remember that there are others, specifically country-based and industry-sector-based ones. I have chosen these examples for useful onward links as well as useful information. I would particularly recommend Stuart Reid's paper comparing test standards [7] and a BSI paper putting a number of software standards into a Software and System Quality Framework (SSQF) [8]. The BSI paper sets the software and systems standards within an organizational model taken from the ISO 9000:2000 [9] family of standards that we looked at in Chapter 1, but it compares this to the EFQM Excellence Model [10] and other quality frameworks, some of which are industry specific. It shows how a framework of standards can be used for the development and delivery of software, and places these within both a technical framework and an enterprise framework. It also sets out a measurement framework, including the QA and QC activities. For QA specialists, whether carrying out process reviews, auditing or making process improvement suggestions, this paper is a useful starting point. Reid's paper concentrates on standards that particularly relate to QC and, within that, software testing. This paper is good reading both for QC practitioners, whether testers or reviewers, and for QA specialists who are auditing or reviewing the test process.

When designing the quality framework for a particular set of measurers, it is essential to consider the quality framework for the customer (Chapter 3). The customer's goals inform their quality framework, and, in turn, this should inform the measurers' quality framework, and also the teams' and individuals' personal objectives and targets.

In Chapter 1, we looked at the EFQM Excellence Model [10] and how it is divided into nine parts: five enabling criteria and four criteria for measuring results. In Section 6.4.2 we will look at how the EFQM Excellence

Model enablers could be interpreted for measurers, and in Section 6.4.3 we will look at measurers' results. Remember that this model is based on the fundamental concepts of excellence we discussed in Chapter 1, and that equivalent models such as the Baldrige model are available.

6.4.2 EFQM Excellence Model enablers for the measurers

6.4.2.1 Leadership

In order that the measurers are supported in their role, the organization will require leadership. As we saw in Chapter 1, project managers provide leadership to their projects, drawing on the lead they are given by the organization's board and management. Measurers particularly require leadership support for their activities, for two related reasons:

- Because the measurers do not build products, their activities may not be seen as "adding" anything.
- Because the measurers are often messengers bringing bad news, their contribution is seen as negative.

This leads to comments like the one I heard from a programmer once: "I don't like testers; all they do is break things!"; and from a senior manager talking about a colleague he hoped would leave (!), "He doesn't contribute anything useful—let's put him in the quality group until he retires." Leaders in the organization and in the project must support the measurers' activities:

- By making the measurers' process and product measures clear to all parties;
- By helping all the groups to agree on a shared quality viewpoint.

6.4.2.2 Policy and Strategy

Policy and Strategy in an organization set down what is expected. For the measurers, an organization would need to have policies that lay down the basic rules for approaching these activities:

- A QA policy, including audit and process review;
- A QC policy, including test, inspection, and product review.

Strategies and high-level plans would be needed for the organization and, perhaps in more detail for particular programs of work or projects, for both QA and QC activities. The strategies and overall plans might include:

- An audit plan for the organization, including the approach, responsibilities, and a schedule of dates for audits;
- A strategy for testing for the organization or for a program, describing the risks to be addressed by testing, the approach, the test stages required in each project, and the completion criteria for the testing;

- A review plan for a particular project, listing the products to be reviewed, the type of reviews required for each product, the goal of each review, and the entry and exit criteria for the reviews.

I have not attempted to give the full list of strategies and plans, nor have I suggested complete contents lists. The specialist measurers should have their framework in place, including tailored standards based on ISO and IEEE standards (see Appendixes A and B for sources). Members of other groups should expect to support the use of those standards.

6.4.2.3 People

Specialist and nonspecialist measurers—skill and aptitude mix. The people who carry out measurement activities need to be selected for interest and aptitude in the area, but will require training in the QA and QC activities. When we examine the measurers group, as well as the specialists, we see people entering and leaving the group, carrying out QA and QC measurement activities. This means members of all groups will need training and support in these areas (Table 6.2).

Aptitude and interest are very important, but if we list what we might expect from people in this group, we see that it is most unlikely that a single individual will have all the aptitudes and skills we require:

- IT skills, for example, analysis and design skills, data modeling, coding;
- Knowledge of software engineering processes;
- Knowledge of QA processes, for example, audit, review, problem root-cause analysis;
- Business/industry sector knowledge;
- Specific software package knowledge;
- Knowledge of QC processes, for example, test design, inspection, review;

Table 6.2 QA and QC Training Requirements Across the Group

Group	*QA/QC Activities That Require Support and Training*
Measurers	Audit process, QA review process, document reviews (e.g., inspection, peer review, walkthrough, test design techniques)
Builders	Document reviews (e.g., inspection, peer review, walkthrough)
	Tests [e.g., component-test design and execution ("programmer testing" or unit testing), integration test design and execution]
Customers	Document reviews (e.g., inspection, peer review, walkthrough)
	Tests (e.g., acceptance-test design and execution
Supporters	Document reviews (e.g., inspection, peer review, walkthrough)
	Tests (e.g., acceptance-test design and execution)
Managers	Document reviews (e.g., inspection, peer review, walkthrough)
	Management reviews (e.g., plan reviews, progress reviews, examination of test-management reports and metrics)

- Test tool experience;
- Project management, planning, scheduling;
- Enthusiasm;
- Attention to detail;
- Tact;
- Firmness;
- Ability to negotiate;
- Ability to generate ideas;
- Ability to work under stress.

And you can no doubt think of others!

For this reason, it is best to build up teams of people with complementary skills. For example, Lloyd Roden, in his 1999 EuroSTAR presentation [11], identified four types of testers, with different styles of work (Table 6.3). These styles are all needed in a successful tester, but one individual is very unlikely to exhibit all these characteristics. So the QA and QC teams, including the test teams, need to have a mix of people.

Using the skill mix effectively. There are good reasons to organize the QA and the QC teams to use different personality traits effectively. This is both to allow individuals to shine in roles that allow them to use their skills and preferences, but also to match the team members' communication styles to their audience. In one organization I visited, the IT quality assurance group decided to have two groups of people in its QA review and audit team. The first group consisted of people chosen because they were known to be knowledgeable and experienced, and who were well liked and had a natural tendency to be helpful. This group carried out process reviews and audits with the project managers, agreed on lists of project risks, and reported a set of recommended improvements to processes. They were characterized as the "village bobbies" (old-fashioned local policemen, whose role was to patrol, advise, and gently admonish). Their objective was to "support projects and implement standards," and the consultants' unthreatening approach meant that projects invited them in. The second group were much fiercer, and their role was to work on projects that had refused the first group's help and were now failing or at risk. They were charged with

Table 6.3 Roden's Tester Type Matrix (Partial)

Pragmatist	*Pioneer*
Efficiency Results Tasks	Change Risks Involving others
Analyst	*Facilitator*
Accuracy Proof Standards	Networking Consensus Status quo

Source: [11].

enforcing processes and escalating problems on projects at risk of failure. The IT quality manager commented:

> One of the key features of our quality approach is recognition that (a) projects are very heterogeneous, thus standards are only a template; (b) we need to empower practitioners to optimize standards. The result is the recognition that an effective IT quality system is dynamic; this means that we recognize both adherence and adaptive quality controls. In one audit, the majority [of deviations from process] were deliberate, based upon considered assessment of the projects' circumstances. Was the project wrong? Absolutely not. When taking on QM, my second-highest issue was that *projects are not tailoring development standards to meet their needs; rather these are interpreted as an inflexible rulebook. This means that projects are not realizing the risk mitigation predicted from adopting a methodology*. Most of [the team's] time is spent selling the benefits of standards, then coaching the practitioners on them (point of need, desk-based) … helping the teams to tailor the standards to their circumstances.

Team organization. One question I am often asked is, "How should the QA and QC teams be organized?" Unfortunately, this is one of those questions that does not have a single correct answer. Some possibilities are shown in Table 6.4. Generally speaking, the greater the risks associated with things

Table 6.4 QA and QC Team Independence

Review Team	*Comment*
Specialists—external (e.g., third-party testers, third-party inspection team, external auditors)	Effective at finding defects against defined process or specification, objective, independent, focused on their customer (may not be project team's customer). Customers and managers will listen to external bodies. May cause fear. May not know enough about the organization. May be seen as an obstacle. Problems may be disguised or hidden.
Independent specialists—internal (e.g., test team or audit reporting to quality director rather than development director)	Effective at finding defects against defined process or specification, objective, independent, focused on their customer (may not be project team's customer), understand the organization's quality drivers, voice on the board. May cause fear. May not know enough about the project. May not understand the project's quality drivers. May be seen as an obstacle. Problems may be disguised or hidden. Quality may be seen as owned by the QA/QC team.
Specialists—advise peer group teams *how to* carry out QA and QC. Carry out spot checks on the QA/QC.	Only a small group of specialists needed. Quality owned by the build teams. Knowledge transfer. May be not so effective or efficient as specialists have to learn new techniques. Builders may take a less objective stance in their QA/QC activities.
Peer group (e.g., walkthrough or test team reporting to the development manager). One project audits another project.	Effective at finding defects against defined process, also at making process improvement suggestions. Understand the problems, "in same boat," allies, will review each other. Less objective. Less independent. Can be difficult to report defects in peer's work. Needs strong leadership and support from managers, and from an independent QA/QC specialist group.
Buddy checks author's work; team leader checks team following processes.	Cheaper than a team, and quicker, but less likely to find defects. Not as effective or objective. Cheap. Team leaders should be "doing this anyway."
Author checks own work	Cheap for low-risk products but often not effective if there are any risks to contain.

going wrong or mistakes being made, the more likely we are to use an independent QA or QC team.

Typically, we find that different levels of independence are required depending on circumstances:

- The higher the risk, the more independent and specialist QA and QC teams will be appropriate.
- At different stages in the project, different levels of independence may be useful. For example, using specialist testers to help the developers design component tests should increase effectiveness. If the developers run the tests this keeps an efficient test run/debug cycle at this stage, which could be followed by independent specialists running integration and system tests.

Motivation and appreciation. When Warden and Nicholson carried out their motivational study [12] (see Chapter 2), one of their key findings was how very demotivated software quality practitioners were. It seemed to be, on examining their roles and their specific job design (the task mix and the opportunities for personal growth), that many of the interviewees had jobs that swung between being excessively stressful and very boring. Key points raised by Warden and Nicholson from the survey results include:

- Testers reported that they used a low variety of skills and could be confined to test execution rather than being able to spend time on test design and problem analysis.
- QA staff reported that they performed too many control tasks and not enough that create improvements.
- QA staff reported they did not get enough direction from management.
- Both QA staff and testers reported they did not get feedback from their jobs; it was difficult to tell if their efforts had made a difference.
- Both QA staff and testers reported that they did not get positive feedback from their colleagues, whether this was people from the customer, builder, manager, or supporter groups, and this was considered to be because they were the bringers of bad news.
- QA and QC staff had lower pay levels and less job security than other IT practitioners.

This finding is borne out by the BCS Industry Structure Model [13], which describes a career progression for various IT roles. The developer roles have a career progression to a higher level than testing roles, yet I cannot think of any real reason why a test specialist, *provided qualifications, aptitude, and experience levels are equivalent to a development specialist,* should not also aspire to an IT director role. Notice, however that QA/audit roles are shown to have a higher progression toward management than QC/testing roles, perhaps because a QA/audit view looks at process rather than specific products, and is focused toward improvement rather than toward defect

identification. I am increasingly hearing testers talk about defect prevention, indicating, perhaps, that there is a move toward QA and away from QC by some individuals. Despite the career progression issues, most quality practitioners I meet, whether QA or QC specialists, are passionate about their work; they may feel unappreciated but they are filled with a belief about the importance of what they do and what they want to achieve.

6.4.2.4 Partnerships and Resources

As we saw in Chapter 1, partnerships cover people outside our team, our project, or our organization with whom we need to exchange information or cooperate in some way. Resources are the things like IT environments, information, and equipment that we need to carry out our work.

For measurers, there are partnerships with other measurement teams, both inside and outside the organization. For example, if I am working as a test manager, I find it useful to meet the audit and compliance teams to see if they have particular areas of interest in the testing. Similarly, I would want to find out what other test managers are doing. This might include discussions with testers working for other organizations on the same project to see where they are concentrating their testing and whether we are competing for resources. I might also look at the output from inspections/reviews and liaise with those teams in the same way. In each case, we are sharing information to ensure that our work complements the other groups' work rather than repeating it.

Resources for measurers include IT resources such as tools and test environments. These are often scarce. I remember arriving on one project at the point at which we had intended to use a particular test environment. Having been assured by the organization's IT and business management that ours was the most critical project and that nothing would stand in our way, we were surprised to find several other projects vying for the environment, all of whom had also been told their project was the most important. I would recommend liaison across the organization as well as within the project to make sure that there is no conflict.

6.4.2.5 Processes

Measurement framework. These are the processes specific to the measurers, not the processes and products they measure. These are the tasks, tools, and techniques that improve software quality by improving the way quality measures are collected and analyzed, and may be divided into two groups:

- QA audit processes, used to check adherence to processes;
- QC review and test processes, used to check products such as documents and code for defects

In the BSI SSQF [8], a group of standards are noted under the measurement framework, including standards for the software measurement process,

audit, software product evaluation, software product metrics, problem resolution, process assessment, and process improvement. You will find standards bodies listed in Appendix A. Standards are important—not only because they should encapsulate best practice, but also because adherence to standards should give our customers some degree of assurance about how we have carried out work. However, they can be quite dry reading, and so you may prefer to refer to a textbook at first. There are a number of good books about QA and QC processes available; you will find a few of them listed in the selected bibliography at the end of this chapter. I am not going to explain the steps through the processes in this book, instead, I am going to concentrate on *how* to carry out the processes in a way that enhances teamwork and quality.

Measurement and improvement. One key point about these measurement processes is that they encapsulate the idea of improvement; not just of the processes and products being measured but also of the measurement processes themselves (Figure 6.2). Measurers must lead by acknowledging their own mistakes and improving their own processes if their suggestions for improvement to other people's products and processes are to be well received.

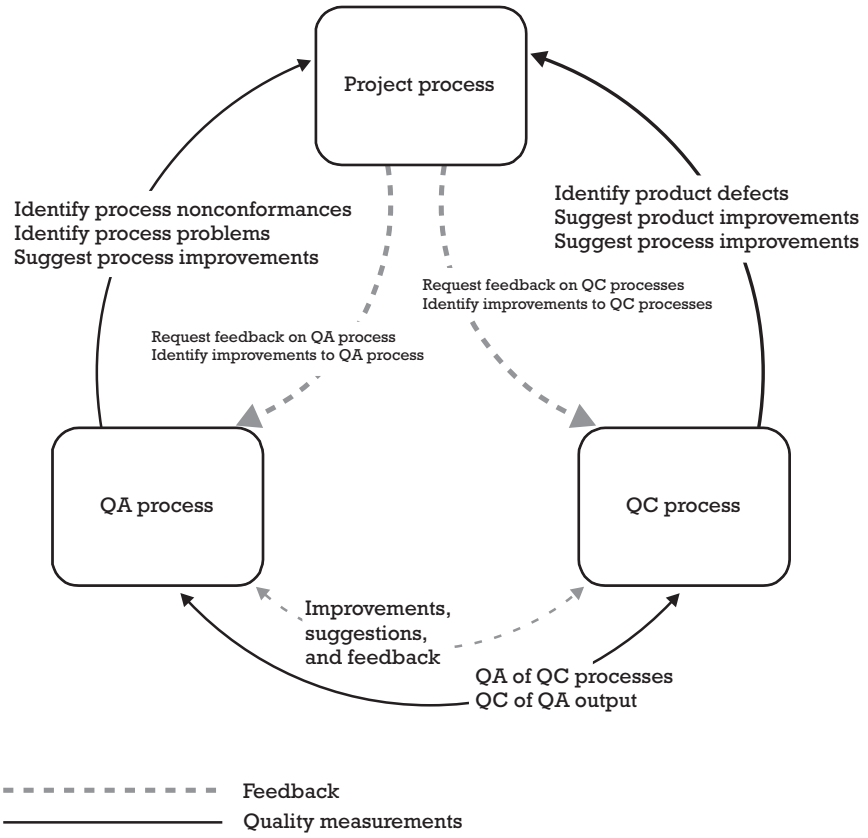


Figure 6.2 Measurement processes include improvement.

Table 6.5 Rules of Thumb for QA and QC Activities

It is the product or the process we are assessing and measuring, not the people	Try not to make it personal; check whether you have any personal baggage you are taking into the activity—you are not supposed to have favorites or be out for vengeance! If you have strong personal feelings that you cannot overcome (such as love or hate), maybe some else should do it instead of you.
Before the activity starts, it should be explained to anyone who is "on the receiving end" and not familiar with the activity	Why, how, who, where, what?—"Hi, I just wanted to introduce myself. Your manager may have said that I'm auditing this project to look at configuration management processes. What I'll need to do is visit you sometime this week to discuss what you do and to look at some examples."
The goals, rules, and plan for the activity should be clearly communicated to all the parties	Why, how, who, where, what?—"As we agreed at the start of the project, I'm going to be auditing your area in the next few days, in order to check adherence to configuration management standards. This is nothing to worry about; what we check is whether the process you use is suitable for your project. We will want to pick up your good ideas to share them with other teams, and we may be able to suggest some improvement ideas to help you. I'll need to interview some people and I'll need to look at the directory contents."
During the activity, collect and classify information, openly	During the activity, the measurers collect information, and this should carefully and clearly documented, differentiating between fact, interpretation, and perception (see Chapter 4). Typically, I make handwritten notes, or you could use a handheld tape recorder, and enter them into the logging tool or type them up immediately on concluding a session or each evening, if it is a several-day activity.
Use the interviewee's jargon, not our own	An example when auditing test processes is that if you ask a nontester, "Do you use the V-model?" they will probably say no, but if you ask them, "Do you review requirements before you start design?" they may well say yes. As people may not know the names of standards and processes, ask them questions about what they do, not what they call it. (If you want to know about the V-model, see Chapter 10.)
Find some positive and some negative things to say	There will always be at least one positive point you can make but, at the same time, everyone appreciates some constructive criticism.
After the activity, give feedback quickly	After the activity, immediate feedback should be given to all the authors and participants on the findings and they should have a chance to comment and add facts, interpretations, and perceptions.
Get feedback from others	Feedback is requested from the authors and participants on the activity—how could it be improved and what would make it more useful to the authors and participants? Facts, interpretations, and perceptions are logged about the activity.
Make sure people know the findings	This might be a detailed catalogue report, an overview management report, or simply access to a logging tool such as is used for test findings, but if the points logged so far are a "good enough" report, I might not write anything extra; I will make sure everyone who needs to know about the finding does know.
Monitor the outcomes—what happened as a result of the activity?	Monitor what actions were taken and why. This generally means follow-up conversations with all the interested parties.
Always ask other people to QA/QC our QA and QC work	"If it's sauce for the goose, then it's sauce for gander," as my grandmother used to say. Showing that we also need help will encourage others to receive help.

Whether we are planning to carry out QA processes such as audit or QC processes such as document review and testing, we should remember some simple general rules of thumb. Table 6.5 shows my rules of thumb. I do not always keep to them (it depends on circumstance), but they are my mental start point and checklist. The specific QA or QC process might be, for example, a document review, executing some testing, or carrying out an audit, so in the table I have called it the *activity* for brevity and to distinguish it from the product or process being assessed.

There are times when these rules of thumb do not apply or apply differently. As we saw above, we may need to follow up some gentle advisory reviews with a harsher audit if risks are escalating and teams are not responding to suggestions. We would also see a difference in emphasis in different types of QA and QC activities:

- In an external audit for compliance to a certification, the timing, rules, and goals for the audit would be set by the external body, and the conduct of the audit is very strict, with minimal feedback.
- In an internal audit or a QA process review, we might want to loosen the rules and allow discussion and information exchange. I have found this especially useful with peer reviews of processes—getting a couple of project managers or business analysts from different areas to talk to each other can be a great experience for both and a fine way to share good practice. I remember that in one organization, the consensus during the audit training was that just doing an ISO 9000-style audit [2] was not helpful; what the teams wanted was to share ideas for improvement, not just to look for compliance to an external standard.

The reporting process. One key process for all measurers is the reporting process. Measurers gather information about processes and products and need to pass this to other people in a way that will be acceptable and understandable. Good information design is vital; it is easy to hide your message so that it is ignored or misinterpreted. However well you have designed your audit or test process, however well you execute the process, if your results are ignored you may as well have not bothered doing it. Reports are the managers' deliverable to all the other groups, and the information that the measurers provide is vital for making these reports an accurate reflection of the quality of services and deliverables. In Chapter 4, I look at some aspects of report design that managers and measurers should consider when setting up reporting for QA and QC. When deciding what to report, do not collect information for the sake of it. One mistake many people make is reporting either on what has always been reported without questioning why, or only reporting on information that is easy to collect. As measurers, we need to reflect other quality views in our reports as we saw in Table 6.1. Also, be prepared to report bad news as well as good news. In my experience, people want to hear the bad news; what they *don't* want is to get it when it is *too late* and without any ideas for improving the situation.

6.4.3 EFQM Excellence Model results for the measurers

6.4.3.1 Customer Results

Customer Results are the measure of the effect we have on our customers. Who are the measurers' customers? In fact, as well as the customers for the software itself, all the groups are customers of the measurers: the managers, the builders, and the supporters. Measurers should look at what the EFQM Excellence Model [10] refers to as customer perception results and customer performance indicators. Perception results are measured externally; we ask the other groups for their views. Performance indicators are measured internally; what actually did other groups do? Both measures are needed so that we see the difference between "we think you are doing a great job" and actually *being invited back* for new projects. In one organization, the head of a new QA group said to me, "I'll know we've succeeded when the managers come to me to ask for audits." Examples of the type of results that could be measured are shown in Table 6.6.

6.4.3.2 People Results

In this case, the People Results measure what the measurers think about themselves. We saw in Section 6.4.2.3 that Warden and Nicholson found two groups within the measurers to be demotivated. The use of a sophisticated and proven measurement technique such as the motivational survey process used by Warden and Nicholson allows a much greater depth of understanding of people's motivation and of how to improve this. As with the customer results, EFQM Excellence Model people results are divided into two groups: the perceptions and the performance indicators. Examples of the type of results that could be measured are shown in Table 6.7.

6.4.3.3 Society Results

For the EFQM Excellence Model, society means society at large, and although we might want to measure this for the organization, we could also interpret it for a particular project as the wider organizational culture within which the measurers work. Again, we consider perception and performance

Table 6.6 Customer Results—What the Other Groups Think of Measurers

	Example of Possible Measures
Perceptions—external to the group—surveys, interviews, compliments, and complaints	Have the measurers got a good image?
	Do other groups report that they find the measurers easy to deal with (for example, are they helpful, flexible, honest, and proactive)?
	Are the measurers seen as reliable, providing good value, providing a good service?
Performance indicators—measurement of outcomes	Do we get asked back in for new projects?
	Are people proactive in coming to us for help?
	Are our reports usually accepted?
	Do other groups complain about our work?

Table 6.7 People Results—What the Measurers Think of Themselves

	Example of Possible Measures
Perceptions—external to the group—surveys, interviews, compliments, and complaints	Has a motivational study provided positive feedback? Are measurers satisfied with their career path, rewards?
Performance indicators—internal to measurers group	Does the measurers group match up to the required qualifications and competencies? For example, do they have appropriate BCS qualifications [14]? What are absenteeism and sickness levels compared with other groups? Are audit staff showing a different absence pattern to other staff? What is the level of staff turnover compared with other groups and do people want to be recruited into this group? For example, if we advertise a document inspection course internally, is it over- or undersubscribed?

indicators. Examples of the type of results that could be measured are shown in Table 6.8.

6.4.3.4 Key Performance Results

The EFQM Excellence Model divides key performance measures into two areas: financial and nonfinancial measures. There was a big debate among delegates at the EuroSTAR Conference in 2002 about how to measure return on investment in testing, and it quickly became apparent that nonfinancial as well as financial costs and benefits needed to be measured. Examples of the type of financial and nonfinancial results that could be measured are shown in Table 6.9. I recently asked a manager in a large IT organization in what ways his testers could improve their services to the organization. His response was that he wanted not just improved technical skills—"more bugs found"—but also an improved understanding of the financial drivers for the organization—"will this make us money?"

Table 6.8 Society Results—How the Measurers Relate to Society

	Example of Possible Measures
Perceptions—external to the group—surveys, interviews, compliments, and complaints	Are the measurers seen by the organization and wider society as acting ethically? Do the auditors and testers check that the systems act properly with regard to health risks, safety, hazards, and the environment?
Performance indicators—internal to measurers group	Has the test group won any accolades, for example, for paper- and energy-saving initiatives, in line with corporate initiatives? Has the audit group checked that certification and clearances with external authorities have been authorized and cleared properly?

Table 6.9 Key Performance Results—Financial and Corporate

	Example of Possible Measures
Financial	What is the return on investment (ROI) for the audits we have carried out this year (cost of audits, number of noncompliances detected, predicted failure costs of noncompliances)?
	What is the cost–benefit ratio of the system-testing activities, for example, cost of running the system test team, against predicted cost of failures if testing had not been done? (The benefit is the *money saved by doing testing*.) Compare with the cost–benefit ratio if testing and other QC activities had been carried out earlier. Consider quality losses.
Nonfinancial	Number of projects that request process reviews, and the size of these projects.
	Cycle times for testing and for document review; how quickly do the builders get feedback on their products and how accurate is that feedback?
	Innovation by the measurers (e.g., new techniques to enable faster implementation of systems to support time to market with new products).

6.5 Communication between the measurers and other groups

We will see in Chapters 8 to 12 that measurers are involved in the whole life span of a piece of software, from its conception, through the software-development life cycle (SDLC), during delivery, and postdelivery until decommissioning. We will see that some of the SDLC models encourage measurer involvement and other models do not.

First, let us think about what it is like to have your work tested, reviewed, or audited. If we think about people's feelings, we can see that communication before, during, and after a QA or QC process is very important. People are often apprehensive if their work is to be examined by other people and, therefore, can become defensive.

Here is a story from a colleague about interrelationships between some measurers and a group of builders.

> One of the main risks that I raised was that mutual distrust between groups—particularly business analysts and the test team—was inhibiting productivity, and until addressed would militate against improvement. The

Table 6.10 Some Measurers and Builders Indulge in Bad-Mouthing Each Other

Person	*Their Comment*
Test manager	"The problems in the project are caused by the poor quality specifications."
Business analyst 1	"The specification defects arise because the testers won't review our specifications."
Tester 1	"I would never speak to a business analyst!"
	(When asked why she did not highlight defects in the specification.)
Tester 2	"It's so frustrating—the analysts think we're a waste of space. It's the way that others carry on.... It reflects badly on me.... They don't see what I do."
Business analyst 2	"The testers are the problem—we have no idea what they do. They are over-resourced and don't have the right skills."
Tester 3	"The business analysts don't do anything—they're lazy."
Business analyst 3	"What does the (lead tester) do? ... plan his holiday, I think."

> comments I got from interview sessions were revelatory [see Table 6.10]. Neither group had any awareness of what the other group was *actually* doing; all were frustrated with the situation.

Is the test manager right? Are the problems in the project caused by poor quality specifications? I do not think so; the poor quality of the specifications is a symptom and not a cause. The problems here are caused by distrust and poor communication. Once test execution had started, my colleague's pessimism about the outcome proved to be well founded: "During test execution, 50% of UAT conditions generated an error and 50% of these were specification defects."

My colleague also observed that the dissonant relationships were a big impediment to delivery:

> The most amazing manifestation of this was in how defects were resolved. The business analysts and testers sat together in an integrated team but didn't interact. Testers entered questions for the analysts in a spreadsheet, for example: *Is this word in the output meant to be capitalized?* Once a week, the business analysts would enter replies into the spreadsheet. As there had been a dispute about the format of the spreadsheet (the testers "won"), several business analysts refused to participate in the process. Directly observed result: simple queries about the specifications took *weeks* to resolve, causing the testing process to slip massively. The fascinating thing was that they had been provided with the environment to work closely (colocated, integrated team), but the relationships were totally broken.

What strikes me most here is how very badly both groups are behaving. Instead of partnership, we have conflict. As we each can only alter our own behavior and as this is a chapter about the measurers, I shall concentrate on their behavior—what could these testers do to improve the business analysts' view of them? Here are some ideas:

- Change the communication system; start talking to the business analysts instead of sending them spreadsheets.
- Ensure that the queries are not trivial; focus on what is important rather than what is easy.
- Be polite to the analysts—they are as busy you are.
- Be seen to work hard and concentrate; stop the "holiday planning."
- Ask the analysts what they need from you; treat them as a customer and provide a service.

Measurers need to share information with all the groups (Figure 6.3). In order to decide where to focus QA and QC activities, measurers need to understand the quality views of all the groups, particularly the customer's goal; the risks—both the impact and the likelihood of problems; and the budget for the project. For example, audit teams will concentrate on high-risk projects, where failure would have a high impact.

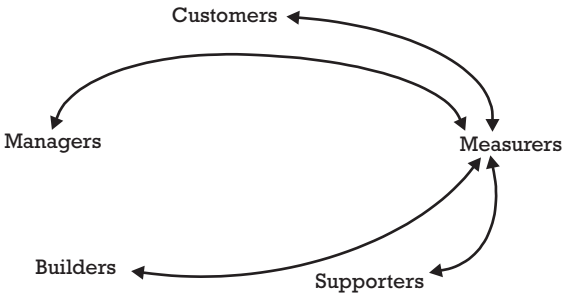


Figure 6.3 Communication between groups.

In Table 6.11, I have identified information that the measurers gather that is needed by other parties. Some of these are measures directly made, for example, numbers of defects; some of them are extrapolations from

Table 6.11 Information That Measurers Have That Others Need

Before the SDLC starts and updated throughout the SDLC	Identification of similar problems/solutions
	Identification and assessment of technical and business risks
	What QA and QC is necessary to address the risks
	What QA and QC is possible within budget constraints
	Advice on methods, processes, and standards, generic and tailored
During the SDLC	Fast and early feedback on the success of the SDLC tailoring, by audit
	Reassessment of risks and consequent retailoring advice
	Improvement suggestions to make activities more effective and efficient
	Defect identification by review of requirements, design, specifications, code, training material, and documentation
	Defect identification by designing, executing, and following up tests of software and processes
	Advice on next steps if stage entry and exit criteria are not met
	Quality measures, including number of defects, risk assessments, cost projections, acceptance criteria passed, projected impact of leaving/repairing defects, including cost, time, and exposure
	Refinements to requirements, acceptance criteria, priorities, risks, constraints
	Advice to nonspecialists and members of other groups
At delivery	Known defects list
	Advice on next steps if stage entry and exit criteria are not met
	Quality measures, including number of defects, risk assessments, cost projections, acceptance criteria passed, projected impact of leaving/repairing defects, including cost, time, and exposure
Postdelivery	Evaluation of risk assessment. Were all the risks identified correctly? Did we have any surprises that we should have anticipated? If any risks were identified and did not become problems, is this because we misjudged the likelihood or whether it just did not happen this time?
	Evaluation of QA and QC processes, efficiency, and effectiveness of quality processes
	Evaluation of management, build, and support processes, improvement suggestions for all processes
	Advice on impact of change
	Aid in using tools and techniques (e.g., regression test packs)
	Aid in using known problem lists and workarounds

those measures, for example, projected costs of repairing the defects and the cost of leaving them; and some are information based on experience.

Measurers need information from other groups, in order to understand the constraints on delivering their wish list. Table 6.12 lists the information measurers need from the other groups.

6.6 Summary of the group

Measurers must communicate well with the other groups if they are to be effective. Indeed, many measurers will be in other groups. The contribution of QA and QC to the success of a project can be enormous, but if the

Table 6.12 Information That Measurers Need from Others

Customers	Who the customers are and their real needs;
	Constraints and risks for the customer;
	Understanding of the customer's user-based quality viewpoints;
	Aims and objectives and acceptance criteria for the project;
	Mandatory requirements;
	Business view of risks (likelihood and impact);
	Quality targets within the constraints;
	Changes to requirements, acceptance criteria, aims, risks, and constraints;
	Number and impact of defects reported during live use;
	Customer satisfaction with the delivered products;
	Improvement suggestions for quality processes;
	Test priorities for the next release;
	Evaluation of QA/QC effectiveness.
Managers	Constraints and risks for managers;
	Better understanding of the manager's value-based quality viewpoints;
	Understand and agree to an outline plan, including constraints such as dates and costs;
	Changes to requirements, acceptance criteria, aims, risks, and constraints;
	Evaluation of QA/QC efficiency from the managers.
Builders	Constraints and risks for builders;
	Better understanding of the builder's manufacturing and product quality viewpoints;
	Understand the technical view of risks (likelihood of errors being made in the build);
	Requests for changes to requirements, acceptance criteria, aims, risks, constraints, and changes to design, code and other products;
	Evaluation of QA/QC effectiveness.
Supporters	Constraints and risks for supporters;
	Better understanding of the supporters' user and product quality viewpoints;
	Understand the technical and business view of risks (likelihood and impact) on existing system;
	Supporters' requirements and acceptance criteria;
	Changes to requirements, acceptance criteria, aims, risks, and constraints;
	Number and impact of defects reported during live use;
	Customer satisfaction with the delivered products as expressed to help desk;
	Supporter satisfaction with the delivered products;
	Evaluation of QA/QC effectiveness.

behavior of measurement specialists alienates the other groups, then no benefits will be realized.

We will see in Chapters 8 to 12 how measurement, whether through QA or QC activities is necessary throughout the life of a software system.

References

[1] Draper, S. W., "The Hawthorne Effect: A Note," http://www.psy.gla.ac.uk/~steve/hawth.html (March 12, 2003), September 2003.

[2] International Standards Organization, ISO 9000: 1994 Quality Systems.

[3] British Standards Institute, BS7925-1:1998, "Software Testing, Part 1: Vocabulary."

[4] British Standards Institute, BS7925-2:1998, "Software Testing, Part 2: Software Component Testing."

[5] International Standards Organization/International Electrotechnical Commission (ISO/IEC), DTR 9126, Software Engineering—Software Product Quality (Parts 1–4, 2000/2001).

[6] Giakatis, G., T. Enkawa, and K. Washitani, "Hidden Quality Costs and the Distinction Between Quality Cost and Quality Loss," *Total Quality Management*, Vol. 12, 2001, pp. 179–190.

[7] Reid, S. C., "Software Testing Standards—Do They Know What They are Talking About?" http://www.testingstandards.co.uk/publications.htm, August 2003.

[8] British Standards Institute, PD0026:2003, *Software and Systems Quality Framework—A Guide to the Use of ISO/IEC and Other Standards for Understanding Quality in Software and Systems*, London, England: British Standards Institute, May 2003.

[9] International Standards Organization, ISO 9000:2000, Quality Systems.

[10] European Foundation for Quality Management, "EFQM Excellence Model," http://www. efqm.org, August 2003.

[11] Roden, L., "Choosing and Managing the Ideal Test Team," *EuroSTAR Conference,* 1999.

[12] Warden, R., and I. Nicholson, *The MIP Report—Volume 2—1996 Motivational Survey of IT Staff,* 2nd ed., Bredon, England: Software Futures Ltd., 1996.

[13] British Computer Society, "Industry Structure Model," http://www1.bcs.org.uk/link.asp? sectionID=574, September 2003.

[14] British Computer Society, "BCS Qualifications," http://www1.bcs.org.uk/link.asp?sectionID=574, September 2003.

Selected bibliography

The American Society for Quality Web site has articles and information about quality issues on http://www.asq.org, including a quality glossary at http://www.aq.org/info/glossary/index.html.

Craig, R. D., and S. P. Jaskiel, *Systematic Software Testing*, Norwood, MA: Artech House, 2002.

Crosby, P, *Quality Is Free*, New York: Mentor, 1980.

The W. Edwards Deming Institute Web site has articles on Deming's work at http://www.deming.org/theman/articles/articles_gbnf04.html.

Dustin, E., *Effective Software Testing*, Reading, MA: Addison-Wesley, 2003.

Galin, D., "Software Quality Metrics—From Theory to Implementation," *Software Quality Professional*, June 2003, pp. 24–31.

Gerrard, P., and N. Thompson, *Risk Based E-Business Testing*, Norwood, MA: Artech House, 2002.

Gilb, T., and D. Graham, *Software Inspection,* Reading, MA: Addison-Wesley, 1993.

Kaner, C., J. Bach, and B. Pettichord, *Lessons Learned in Software Testing*, New York: Wiley, 2002.

Pol, M., and E. van Veenendaal, *Structured Testing of Information Systems*, Deventer, the Netherlands: Kluwer, 1998.

Spaine, S., and S. P. Jaskiel, *The Web Testing Handbook,* Orange Park, FL: STQE, 2001.

The Sticky Minds Web site has roundtable sessions has a continuously changing range of articles and discussions on testing and software quality issues on http://www.stickyminds.com .

All issues of the *STQE Journal* (about to become *Better Software*).

Watkins, J., *Testing IT: An Off the Shelf Software Testing Process*, Cambridge, England: Cambridge University Press, 2001.

Wilborn, W., "Dynamic Auditing of Quality Assurance: Concept and Method," *International Journal of Quality and Reliability Management*, Vol. 7, No. 3, 1989, pp. 35–42.

CHAPTER

7

Contents

Roles and Quality: Supporters

In this chapter I shall:

- Introduce the members of the supporters group, their roles, and activities;
- Introduce their quality viewpoint;
- Provide a framework for the supporters activities within the EFQM Excellence Model;
- Identify information flows between the supporters and the other groups.

So the first we hear about the new system is when we are asked to give it an operational acceptance test, just before the project team plan to put it live, so there's no time to do anything much. We get delivery of the software and run a quick test of the overnight batch; it now takes too long. Turns out the development guys and the testers never asked Operations about what constraints are on the time slots. That meant a last-minute fix! I didn't realize you didn't know about it.... I'd have told you if I'd realized. Yeah, we knew it was coming, but we didn't realize the mess they'd made of the user interface. The help desk is inundated with calls, and the extra network traffic is way over the capacity we'd planned for it. They should've come to talk to us; we'd have put them right. Now we've got to pick up the pieces.

—Some supporters moan about the newly implemented software

7.1 Introducing the supporters

Once the software has been and built and delivered, the customers will use it. This postdelivery period in the life of software, described in Chapter 12, is the longest period in the life of software, and its most important time; after all, if the software does not have a postdelivery life, what use is it?

Postdelivery, a group of IT specialists look after the software and the customers, providing a number of support and IT infrastructure services. These people provide the management, support, and infrastructure for the deployment, update, optimization, and use of the software. Additionally, this group plays an important role during the software-development life cycle (SDLC), providing the infrastructure and support within which the products are built as well as support for specialist tools used by all the other groups. They also keep "business as usual" going on the IT systems during the SDLC and the delivery.

Because of their role in supporting the software, this group has a breadth and depth of knowledge unrivaled by any of the other groups. They see the software in use and in context with the rest of the system. They work with the software from its delivery to its decommissioning, supporting it and its users. They are the first to hear about its faults from the user community, and they will be the people who fix those faults. I have chosen to call this group the *supporters* because they support all the IT-related activities of the other groups and, hence, the organization as a whole. They are critical to the success of the software and of the organization as a whole. Why, in our story at the start of the chapter, are the supporters so fed up? Well, you cannot provide a good supporting service unless the software is supportable, and the delivered software is unsupportable. They have information that would have helped the builders and measurers deliver better software. They have service needs that have not been met. They are on the receiving end of complaints about the software from the customers, but they were not involved in its designor build. They may even be subject to penalties against service-level agreements that have been compromised by the poor performance of the software.

Too often, in my experience, the supporters are involved too little and too late in the SDLC. So, why have I left them until last in this group of chapters? It is to emphasize their importance. Builders (Chapter 5) are central to this group of chapters and to the delivery of products, including software. They are aided in their work by managers (Chapter 4), who are conduits for information, and by measurers (Chapter 6), who provide information about the products and processes. All this work is done for the customers (Chapter 3); without them we would not need or build the products to support them in their work. Without the supporters, the customers might not be able to use the software. This is because customers may not have the technical knowledge, interest, or time to support the software themselves, for example, when:

- Software has to be deployed on a complex infrastructure of hardware, communications, and systems software.
- Software has to be protected against infiltration, losses, and other security risks, as does the information stored with it.
- Communications and on-line and batch-processing systems have to be monitored, and sometimes nursed, day and night.

- Software and its infrastructure require updates and changes, which need to be deployed without adversely affecting business as usual (BAU).
- Complex software is not learned quickly; the customers need the support of a help desk as they use more of the software's features.
- The software is optimized during its life and use [1].

The supporters have knowledge, experience, and information that none of the other groups have. Just like the customers, they have requirements and acceptance criteria for the software, which will enable it not simply to be functionally correct and provide the features the customers need, but also to provide quality in use [2], by having appropriate attributes of performance, security, reliability, and so on. So, the customer and supporter chapters "bookend" the chapters about the people they need to inform. Just like the customers, the supporters need to be involved right from the start of planning the software, through the SDLC, as well as during use and deployment. Many of the points about customer contact throughout the book also apply to supporters.

In this chapter, we will see that of all the groups, the supporters is the one that is most mature in its understanding of the importance of IT to the organization, and has standards that acknowledge and depend on wider organizational excellence frameworks. We will see in Chapters 8 to 12 that the involvement of the supporters is vital throughout the entire process of planning and executing the SDLC, not just at the point of delivery and after, because of this understanding of the attributes and the activities necessary to provide the IT services required by the organization and the customers.

7.2 Who could be in this group?

Supporters carry out all the IT service management activities required to support the organization. Generally, these jobs will be done by specialists. It is likely that the team that looked after the IT infrastructure, communications, and networks would be different from the team that manned the help desk. The specialists we might encounter include:

- Service-support specialists, including people who deal with the service desk, incident management, problem management, configuration management, change management, and release management;
- Service-delivery specialists, including those who work in capacity management, financial management for IT systems, availability management, service-level management, and IT service-continuity management;
- Information and communications technology (ICT) specialists, who are specialists dealing with network-service management, operations

management, management of local processors, computer installation, and acceptance and systems management;

- IT security specialists, who will deal with internal and external security of the IT systems and the information held within them;
- Software-maintenance specialists, including those who deal with systems and data conversion, as well as those who correct and enhance software.

Specialist supporters are important stakeholders for quality. There are two parts to their involvement:

- First, this group maintains the software when it is delivered and accepted by the customers; they provide support and infrastructure and are therefore stakeholders for quality in that they will have requirements for the software, particularly in its quality attributes. Security, performance, portability, and maintainability will all be factors for this group.
- Second, this group affects the quality of processes and products because they provide the support and infrastructure for the other groups during the SDLC. They will supply the environments for the building and testing of the software and will support the tool sets used by all the groups in their work. The group includes the IT infrastructure team, comprising IT operations, support, and maintenance; IT security; the help desk; service management; networking; and database administration (see IT infrastructure staff and software maintainers in Chapter 2, Table 2.1).

Many of us get involved in these activities in some way. Anyone who writes documents has to deal with change management and version control for those documents, for example. In smaller organizations, people may move between groups, sometimes building and supporting the software systems. Indeed, it could be argued that most software engineers work in software maintenance rather than software development. Most projects are about altering existing software rather than building entirely new systems. Similarly, in some organizations there may be an overlap between the measurers and supporters. In several organizations, I have seen the system testers also manning the help desk. This can be useful, as the testers develop their understanding of the customers' real use problems, but I have seen the situation were the team is overwhelmed by the volume of testing and help-desk calls for which they are responsible.

7.3 Quality viewpoint

Supporters need the software to be supportable; it needs to be fit for purpose and to have attributes that make it supportable. It needs to fit within infrastructure constraints and operational profiles. Remember our supporters at

the start of the chapter—if the batch cannot run in its assigned slot, later processes are affected, and this may result in the on-line system not being available on time for the customers. Supporters tend to be more aware of organizational requirements that affect the wider organization than other IT staff, because they come up against problems more directly. I remember seeing a payroll system in which the overnight batch run for the weekly paid staff took somewhat longer than previously. It would not fit into the overnight schedule on Wednesday night, so someone in the development group suggested it could run in the freer Thursday night schedule. Weekly workers were either paid into their bank accounts, or by cash. Running the batch Thursday still left time for the head office weekly paid staff to get their cash envelopes by lunchtime as usual. What the head-office-based, monthly salaried development staff did not consider was that the manual workers for the organization worked remotely from head office, some as many as 50 miles away. This was why the payroll ran Wednesday; it left time to put cash into envelopes and take it to the remote sites, to enable the manual workers to be paid at lunchtime on Friday. If the payroll was run on Thursday, these workers would not receive their money until Monday, unless the courier could leave by 10 a.m., meaning that the cash clerks would need to start work at 7 a.m. instead of 9 a.m. every Friday. What implications does this have for the organization? It is an unplanned change in working practices. We would need to consider the reaction of the workforce and unions, the effect on management and the human resources department, adverse publicity in the press, possible strike action, and so on. Yet the IT development manager said to me, "Why does the time the courier leaves the pay office affect us? That's not an IT issue!" Yes, I am afraid it is, because the working of the IT system is affecting the organization and its staff. It was the *supporters* who understood and pointed out the problem, because they knew the special provisions that would be required if the Wednesday payroll failed in the existing system and it had to be run as one-off process on a Thursday night.

Supporters favor the product-based definition of quality, based on a well-defined set of software quality attributes that must be measured in an objective and quantitative way. We can derive acceptance criteria to objectively assess the quality of the delivered product. The supporters need this objective measurement of attributes to help them ensure that they will meet the service levels expected by their customers, the IT users, and the organization. For example, standards such as ISO 9126 [2] define quality—in-use attributes that directly affect the supporters' work. Typical attributes are reliability, usability, security and maintainability.

Supporters also favor the user-based definition that says that quality is fitness for use. This is because they will be using the software, as operations teams, for example, but also because they are in the first line for dealing with user inquiries and customer complaints. In the user-based view, software quality is determined by the user(s) of a product in a specific business situation. Different business characteristics require different "qualities" of a software product. For example, the usability of the interfaces for operations such as daily updating of security measures against virus and spam attack,

running the daily batch systems, back-up and recovery, installing and deinstalling software, will all be of interest to the supporters, because they are the users of those programs.

The supporters' managers will focus particularly on service levels. This may include a provision for the cost of supporting the systems. In Chapter 12, we will meet a disenchanted support manager caught by service costs rising with a new software release. Failing to meet the service levels may incur a penalty clause. This means supporters and their managers have an interest in the value-based view of quality, specifically whether the money saved or spent during the SDLC is justified by savings in use of the system. We will look in Chapter 12 at evaluating postdelivery.

This group's transcendent view of quality—their taken-for-granted assumptions about the systems—include notions of stability, reliability of service levels, and long-term benefits. These assumptions about quality reflect the supporters' need to carry out their own work efficiently and effectively, as well as supporting the customers' use of the software to meet the organization's ends.

We can see in Table 7.1 how supporters, in this example a help-desk team, could consider the other groups' quality viewpoints to help them understand the supporters' quality requirements.

7.4 Quality framework using the EFQM Excellence Model

7.4.1 The EFQM Excellence Model and the supporter

As I mentioned earlier, the supporters have a mature standards framework. This is supported by a global organization, the IT Service Management Forum (itSMF) [3] which supports an associated library of process descriptions and guides, including the IT Infrastructure Library (ITIL). This library has been developed since 1989 [4], has recently been updated, and is being released as a series of seven guides. The ITIL books set service management within an EFQM Excellence Model [5] framework, and also discusses the similar Baldrige [6] framework used in the United States. The subjects in ITIL are:

- Service Support—covering service desk, incident management, problem management, configuration management, change management, and release management [7];
- Service Delivery—covering capacity management, financial management for IT systems, availability management, service-level management, and IT continuity management [8];
- Security Management—covering fundamentals of information security, relationship of security management to other aspects of service management, and security management measures [9];

Table 7.1 Help-Desk Team Considering Other Viewpoints for Quality Impacts

Group and Quality View	*Possible Routes into a Discussion with This Group*	*Example Questions and Areas for Discussion*
Customer User-based quality view	Shared viewpoint—discuss the impact on *fit for purpose* if things go wrong	Impact on business as usual (BAU) of failure, for example, cost of system outage to the business, work throughput efficiency for customer.
Builder Product- and manufacturing-based quality view	Shared product view; product attributes	Which nonfunctional areas has the customer discussed with you?
	Show empathy for manufacturing view; technical excellence and defect likelihood.	These are the nonfunctional attributes that experience shows us are important for BAU—the attributes most often commented on by customers calling the help desk.
		Volunteer to review specifications/requirements for possible attribute problems (functional and nonfunctional).
		Provide list of common help-desk issues and discuss technical innovations that could help improve these areas.
Measurer Product- and manufacturing-based quality view	Shared product view; product attributes	Priority of tests and document reviews. These are the nonfunctional attributes that experience shows us are important for BAU—the attributes most often commented on by customers calling the help desk.
	Show empathy for manufacturing view; defect likelihood and impact.	Volunteer to review and comment on test designs and test results; these are our acceptance criteria for product attributes.
		Discuss how to assign priority for defects, based on impact and likelihood for BAU.
		Provide support for setting up like-live environments for tests; provide example tests.
Manager Value-based quality view	Identify the *cost* of things going wrong and the *cost* of repair/support.	Service-level agreements and their penalty clauses, shared risk on penalty clauses.
		Cost of support, ROI on maintenance changes.
		This is the cost of system outage.
		This is the cost in customer and supporter time/efficiency loss in help-desk calls.

- The Business Perspective (was scheduled to be due in 2004)—covering business relationship management, partnerships and outsourcing, continuous improvement, and exploitation of ICT for business advantage [10];
- ICT Infrastructure Management (not yet available)—covering network services, operations, local processors, computer installation and acceptance, and systems management [10];
- Application Management—covering SDLC support, testing of IT services, and business change [1];
- Planning to Implement Service Management—a "how to start" guide [11].

Along with and aligned to ITIL, standards and qualifications are being or have been developed [12, 13] to cover service management.

Supporters may provide services to the customer as in-house suppliers or as third-party suppliers (see Chapter 3). The supporters' ability to recognize and adapt to the customer's quality framework is a critical success factor in the effectiveness of their relationship, whether they are internal or third- party suppliers. The framework for the customer translates into departmental goals and, finally, into the personal objectives and targets for an individual system user, but it also translates into a complementary subset of the service management supplier's goals and, hence, into service-level agreements (SLAs). These will lead to personal objectives and targets for all the supporters. This quality framework will drive what the customer needs from the software and from the SLAs.

In Chapter 1, we looked at the EFQM Excellence Model and how it is divided into nine parts: five enabling criteria and four criteria for measuring results. In Section 7.4.2, we will look at how the EFQM Excellence Model enablers could be interpreted for supporters, and in Section 7.4.3 we will look at supporter results. Remember that this model is based on the fundamental concepts of excellence we discussed in Chapter 1 and that equivalent models such as the Baldrige model are available.

7.4.2 Enablers for the supporters

7.4.2.1 Leadership

The supporters' activities are vital to the success of the organization, not just during BAU, but also during periods of change to the organization and IT provisions, and in enabling the organization to deal with threats to business continuity.

It is easy for the organization to take the supporters for granted; sometimes the only feedback they get is when things go wrong. For this reason, it is particularly important that the organization's leaders take account of service management in recognition schemes as well as in reward schemes [8]. For example, if the operations staff keep the overnight batches running on time through day-to-day problems, no one outside the group may notice, unless their leaders can alert senior management and, hence, the organization's leaders that a bonus (reward) and a thank you (recognition) is needed [14]. Supporters may also have to "pick up the pieces" after an unsuccessful software delivery, specifically in dealing with customer complaints and in fixing operational problems. We will meet in Chapter 12 a supporter reeling under the strain of looking after software delivered by a different team.

Similarly, it is easy for business managers to neglect areas that do not resolve immediate problems. Ensuring proper IT service continuity management plans [8] requires strong leadership and a strategy agreed to at the top level and implemented and communicated through the whole organization.

An interesting point raised by Warden and Nicholson in their motivational survey [15] is that the introduction of improved processes, for example, the adoption of the ITIL processes, while providing a process improvement, in some cases lowered staff motivation when jobs were changed radically without good communication from leaders. This is an

important point for leaders. My observation is that professional staff in all groups will want and expect the adoption of appropriate recognized standards, but they do not respond to imposition of standards, especially without suitable training.

7.4.2.2 Policy and Strategy

The organization will need policies and strategies for all aspects of service management; these briefly set out the overall rules for the supporter team's work; for example, an organization might have a release policy [7] that defines release definition (for example "major releases are numbered v1.x, v2.x and must include significant changes to critical attributes of the system"), required deliverables, preferred timing for implementations, or times to avoid (for example "implementations always take place at least two weeks before month/year ends").

These policies must be agreed on and clearly communicated, whether they are documented (preferable in a large organization to ensure they are not miscommunicated) or passed on by oral tradition: "We always do it this way."

The policy rules will be translated into strategies for the supporter's activities. These will show the approach that the supporters will take to meet the goals of the organization. The different teams within the supporters group—IT Infrastructure, help desk, the software maintenance teams, and so on—will have strategies and detailed plans for their own work. Of course, it would be easy for these strategies to become misaligned with each other and with the strategies of the other groups, such as the builders. The ITIL books [e.g., 7] use a nice analogy—that of tectonic plates. The areas of interest and control overlap and demarcation lines are not clear. It is at these overlaps that friction occurs. We have seen in Chapters 3 to 6 that the other groups each have policies and strategies. Now, even if all the groups work for the same organization, some considerable effort in communication is needed to make sure each group understands the others. Policies and strategies need to be discussed, agreed on, and mutually understood, the overlaps of demarcation particularly being a focus for communication. The supporters' policy and strategy must support the customers' policy and strategy, but they also must support the managers, builders and measurers. If some or all of the supporters are from third-party organizations, the communication during agreement on the contract and SLAs as well as throughout the relationship will need to be nurtured across all the groups and across organizations, as we saw in Chapter 3.

7.4.2.3 People

The supporters group includes a variety of specialists, who may be working in different teams and through separate management structures. They may also work in a separate management structure from the builders and measurers. This isolation of teams can lead to one of the problems our supporters

at the start of the chapter experienced: poor or nonexistent communication leads to ineffective, inefficient service delivery to the customers.

Increasing specialization also means that individual supporters may only see part of each problem or solution—work is divided up and passed along from one team to another. If little or no feedback is received by the supporters on the effectiveness of their work, this can be alienating and demotivating, just as we saw it was for the builders in Chapter 5. One finding of Warden and Nicholson [15] was that the feedback received by supporters was mainly negative, but was about the quality of the software—something they typically had no control over—rather than about their own service levels. Typically, an SDLC team of builders and measurers deliver the software and other products. The customer uses them, and when things go wrong complains to a supporter (e.g, one on the help desk). Thus, the supporter does not get feedback on the level of service *they* provided but on the poor product provided by another team. The customers do not ring up to say that the product is good, and when there have been problems they will not be in a mood to comment on the good aspects of the supporters' work. The efficient and effective running of the IT infrastructure, communications links, background processes, and so on is something we tend, as computer users, to take for granted. We only notice them when they go wrong, so the supporters receive poor feedback and, as a result, can become demotivated.

In addition to the problem of feedback, there can be a problem of status. The other groups may undervalue the supporters. I remember some years ago interviewing a software engineering graduate who explained that he did not expect to do any software support or maintenance because that was "boring, low-skill" work. For him, software engineering meant developing brand new systems from nothing, and then—in my view quite, bizarrely—never needing to change them or to support the customer. Instead, "in the unlikely event that anything was wrong with the software" or if the customer "changed their minds about what they wanted," a new system would be built. We had quite a painful conversation about the reality of life with software; the frequent problems and misunderstandings during the SDLC, and the need for constant change to reflect the customers' constantly changing world. In my own view, the work of supporters should be seen as high status; it is not only critical but also extremely difficult, requiring both technical and personal skills. For example, one of the central points of application management as described by ITIL [1] is that software in use will be optimized throughout its life. Changing software, as we will see in Chapter 12, is very difficult; the supporters' role in maintaining and optimizing the systems in order to meet the customers' changing needs and to improve service to them should not be undervalued. Some SDLC models—for example, XP [16] and Evolutionary Delivery [17]—acknowledge, indeed expect, change. We might even see these as software-maintenance life cycles (SMLCs) rather than SDLCs, and the people delivering via these life cycles as supporters rather than builders, software engineers as maintenance staff rather than development staff. Could it just be the poor perceived image of software maintenance that prevents this?

A breadth and depth of technical skills are also required to understand and be able to deal with the range of technology used by the customers, and to assess the technical and business implications of changes in technology. A recent example of change is the increased IT and commercial security risks posed by organizations requiring desktop machines to have Internet access as well as access to internal systems.

Supporters need a mix of skills. Their education, training and work experiences will, of course, include all the technical skills and understanding of the methods, techniques, and tools for their tasks. Just as for the builders, it should also include an understanding of:

- The *measurement techniques* they will need to apply to their own work and to other people's work. For example, supporters carry out QC activities such as reviews and testing, specifically the Operational Acceptance Test (OAT) in which they will need training. Supporters also employ measures that will be needed for the ongoing management of SLAs, and toolsets for providing these. An example might be the network traffic our supporters are discussing at the start of the chapter. They will have an optimum traffic volume calculated and be measuring against that. Too little traffic and we have a network that is not cost-effective: too much and the traffic takes too long. Supporters like the maintainers and the help desk will be interested in measures of products on delivery. For example, we can measure code complexity and use it to understand the difficulty of future maintenance, while measures of the usability, reliability, and stability of the software will give the help-desk team a view of how busy they may be after delivery.
- The *management skills* they require for control of their own time and work within a team. Supporters need the skills, experience, and support to assess risk, negotiate between parties, and manage against SLAs. The focus of ITIL and the itSMF (for example, [1]) is on management. The management skills of supporters need to include an understanding of organizational issues, providing input to the business IT strategy, IT governance, IT risk management, and management of IT and systems to support business change. Recent publications, for example, [18, 19], emphasize the increasing importance of these areas.
- The *interpersonal skills* (sometimes called soft skills) to improve their communication with all the groups, including the other teams of supporters, and to deal sympathetically with people reporting problems to them. One unfortunate thing I have seen is the frustration of some support-line engineers with what they see as stupid questions: "If they just looked at page 123 of the manual, they would find the answer." A good supporter empathizes with the customer's view—maybe if the user can't find the answer, the manual is too complicated or has been written in IT jargon rather than the user's language, or possibly has been badly translated if the software and manual are to be used in many

countries or the supporter's first language may not be the customer's first language.

- We said above that part of the management skills supporters need include strategic planning; this must fit with their customer's *goals and strategy* so that they can focus their technical, measurement, management, and interpersonal skills toward solving the organization's real problems. The links between the customers and the supporters should be very strong.

7.4.2.4 Partnerships and Resources

Supporters operate in a network of partnerships within and outside the organization. These partnerships are in place to help provide the resources managed by the supporters, including the hardware, communications, systems, and software, and the data and information that provide knowledge.

Examples of supporter partnerships are:

- Between groups: with the builders, customers, and measurers to define acceptance criteria for nonfunctional attributes such as reliability, security, performance, to define the problem to be solved, and share information about risks and the fit-for-purpose, user-based quality view.
- Between groups: with the managers to discuss service-level agreements, costs, and value.
- Between groups: with the measurers to understand how operational acceptance criteria can be measured during testing.
- Between groups: with the builders and measurers to understand what resources and environments are required during an SDLC; also with the help-desk team and the trainers to discuss likely training needs and training success.
- Within the supporters group: for example, our supporters at the start of the chapter would benefit from sharing plans and news regularly.
- Outside a particular SDLC, to understand possible conflicts in resource requirements across several projects, or conflicts between the SDLC and BAU.
- Outside the organization: for example, with hardware and equipment suppliers, to plan delivery schedules against capacity and demand forecasts.

7.4.2.5 Processes

Many of the points made in Chapter 5 about builder's processes also apply to supporters. However, one advantage is that there is a globally acceptable definition of the required processes and management for their work in the

IT Infrastructure Library (ITIL) [1, 7–11]. These volumes provide process descriptions and flowcharts, as well getting-started guides and context descriptions of quality standards and excellence frameworks. They also cross-reference each other well. These process descriptions need to be adapted by particular organizations to fit their own requirements.

It is also important to realize that the introduction of new processes will require management of that change. Introduction of new processes or changes into existing working practice can cause motivation problems [15], especially if imposed without consultation and training. It is always worth asking the technical experts to design the processes, based on published standards and their own experience of what is appropriate, then include less-experienced people in the reviews both of the processes and of training material for the processes.

Remember, too, that people do not always follow the processes. Sometimes this means that the process is unworkable; perhaps it cannot be followed in the available time. Sometimes it means they have not understood the reason for the process. I remember one place I worked in many years ago where a small team of builders and measurers working on a minicomputer also had to be their own supporters in certain aspects; we took turns carrying out the daily backup, for example. The process for this was quite clear and written down: a number of tapes were circulated through the daily and weekly backups so that on each day the oldest backup was overwritten. Thus, a set of backups over the previous two weeks was always available. A log was kept of the backups, recording who had done them and so on. I went into the machine room one day to collect something and found the senior programmer, Simon, looking at the tapes in amazed despair. A series of backups had been made over a number of days without circulating tapes properly. We had lost *this week*. as it had been overwritten by *today*. If the computer had failed before today's backup, a week of work would have been lost rather than just today's work. The person who had done the backup was not being deliberately malign, they just did not realize the implication of what they had done. Simon ran a training session explaining not just what must be done but *why it was important* to do it following the process. People are not stupid, and one of the ways they manifest intelligence is by seeing how to "improve" processes. The problem is that if the reason for doing something is not clear, an optimization may remove a safeguard and increase risk beyond what is acceptable.

7.4.3 Results for the supporters

7.4.3.1 Customer Results

All the groups are customers of the supporters because they support the IT systems and software used by all the groups. Their customers will each measure them against their own quality measures—implicit or explicit—and the supporters need to measure their performance for their customers considering those quality viewpoints when setting service-level agreements with customers. We are interested in both the customer perceptions and the actual

performance against service-level agreements. Table 7.2 shows some of the possible measures.

The supporters are in a position to collect customer results not just for their own services, but also to allow evaluation of the builders' and measurers' work done during the SDLC. This should be agreed on and used in collaboration with the other groups.

7.4.3.2 People Results

Here, the supporters measure themselves. What is their own reaction to their work and their services? Again, we will measure perception and performance. There are a number of factors that make supporters roles difficult:

- The work can be very reactive, for example, on a help desk or support line, and is therefore paced not by the person but by the frequency of calls. This makes it difficult for people to plan their time and tasks.
- The work is divided between many interlocked and overlapping processes, with many communication points, so it can be complex to control, for example, with ITIL. If we look just at the Service Support book [7] it describes 17 processes, and the Service Delivery book [8] describes another 16 processes; these may be being carried out by one or many teams depending on the size of the organization.
- The quality and attributes of products and services being supported depend on the product delivery team, who may not be involved in the support of the products and services. This means that the supporters do not have control over the products' quality and yet are affected by it.
- The work may involve unsocial hours and shift work because the infrastructure and systems may be in use and need to be supported 24 hours a day. This may involve tedious waiting interspersed with emergencies, which means that concentration cannot be allowed to lapse. I remember visiting some computer operators in a windowless darkened room filled with monitoring screens that had to be watched continuously for abnormal changes in network traffic. It was not a natural or ideal environment for a human.

Table 7.2 Customer Results—What the Other Groups Think of Supporters

	Example of Possible Measures
Perceptions—external to the group—surveys, interviews, compliments, and complaints	Have the supporters got a good image? Do other groups report that they find them easy to deal with (for example, are they helpful, flexible, honest, and proactive)? Are they seen as reliable, providing good value, providing a good service?
Performance indicators—measures of outcomes	Have we met service-levels agreements for all aspects of service to our customers—service availability, response times, help-line query responses, closure rate on problems, incidents, issues, recidivist problems, service and unavailability?

For these reasons, careful monitoring of people results for the supporters is important, as is a related rewards and recognition scheme. Examples of the type of results that could be measured are shown in Table 7.3.

7.4.3.3 Society Results

For the EFQM Excellence Model, society means society at large, although we could interpret it as meaning the wider organization. Again, we consider perception and performance indicators. Examples of the type of results that could be measured are shown in Table 7.4.

Table 7.3 People Results—What the Supporters Think of Themselves

	Example of Possible Measures
Perceptions—external to the group—surveys, interviews, compliments, and complaints	Has a motivational study provided positive feedback? Are supporters satisfied with their career path, rewards?
Performance indicators—internal to supporters group	Does the supporters group match up to the required qualifications and competencies? For example, do they have appropriate BCS qualifications [13]? What are absenteeism and sickness levels compared with other groups? Are overnight operators showing a different absence pattern than IT security staff? What is the level of staff turnover compared with other groups and do people want to be recruited into this group? For example, if we advertise a help-desk post internally, is it over- or undersubscribed by internal applicants?

Table 7.4 Society Results—How the Supporters Relate to Society

	Example of Possible Measures
Perceptions—external to the group—surveys, interviews, compliments, and complaints	Are the supporters seen by the organization and wider society as acting ethically? Do the auditors and testers check that the IT Infrastructure and systems act properly with regard to health risks, safety hazards, and the environment? This might include checking that the delivered IT service complies with legislation on data protection, availability of services to people with disabilities, or pollution control.
Performance indicators—internal to builders group	Has the group won any accolades, for example, for paper- and energy-saving initiatives, in line with corporate initiatives? Has the group been recognized in its efforts to reduce security risks internally and externally, including customer data confidentiality, prevention of spam and virus attacks, and IT governance provisions? Is there any consideration in the SLAs of the effect of IT on society, environment, and people? Are there links across the organization so that the management of IT systems, data, information, and knowledge is part of an integrated management system that includes, for example, the overall security, environmental, quality, and financial management?

7.4.3.4 Key Performance Results

As we have seen in earlier chapters, the EFQM Excellence Model divides key performance measures into two areas: financial and nonfinancial measures. Some examples are shown in Table 7.5. It is important to measure the supporter's services against the needs of the organization. For a business, this will mean that the key performance results must reflect the commercial drivers for the organization. For a nonprofit organization, again, the results must support the organization's goals, increasing the efficiency and effectiveness of the services and products provided.

7.5 Communication between supporters and other groups

We will see in Chapters 8 to 12 that supporters are involved in the whole life span of a piece of software, from its conception, through the software-development life cycle (SDLC), during delivery, and postdelivery until decommissioning. We will see that some of the SDLC models encourage supporter involvement and other models do not.

We have seen that supporters communicate with all the other groups, but that they are often reactive in communication by the nature of their work. If this is the case, perhaps the supporters should instigate communications with the other groups so they can find out what is planned, and can become part of the SDLC. As with the customers, the advantage of having little involvement to us as supporters is that while it is being built and delivered we get on with other things. Superficially, it looks as though we are being efficient with our time. The disadvantage is that during the period of the SDLC, things will change; our customers' problems will change, it may be that the underlying infrastructure will change, and, therefore, the solutions we require will change. This means by the time we get the software, it is out of date. The disadvantage of SDLCs with high supporter involvement is that it is time-consuming when we have business as usual to deal with as well as the SDLC. The advantage is that it is far more likely that a useful software solution will be delivered. So we will see in the later chapters that communication between the supporters and the other groups is needed

Table 7.5 Key Performance Results—Financial and Corporate

	Example of Possible Measures
Financial	What is the return on investment (ROI) for the services we have delivered this year (cost of services/money saved/number of new customers/market share gained)?
Nonfinancial	Number of projects/customers who request our team to carry out IT-related work, and the size of these projects
	Cycle times for responding to customer requests and questions
	Innovation by the supporter (e.g., new technology to support time to market with new products)
	How closely have SLAs been met or exceeded?
	How well did the agreed-upon SLAs match the goals of the organization?

throughout the whole life span. The supporters need to communicate with all the other groups (Figure 7.1) because they have information that those groups need.

We will see in Chapters 8 to 12 that from before the SDLC starts until the software is decommissioned, the other groups need the supporters. As things in and around the project develop and change, this may affect the SDLC. I remember a project in which we had decided to focus much attention during system test on performance testing, because the performance of the system after change was assessed as the greatest risk to the success of the project. Part way through the system test, the live environment underwent a hardware upgrade that enhanced the performance of the live systems. It became apparent that our fears about performance had been resolved by this upgrade. We reassessed risk, and as a result discarded our strategy of performance testing, concentrating our efforts on other aspects of the system that were now at higher risk than performance. Without information from the supporters, we measurers would not have known about the environment upgrade, as our environment was not being upgraded, so we would have wasted effort instead of concentrating on the real risks. Table 7.6 lists some of the information that supporter have that other groups need.

Supporters need information from other groups in order to understand the constraints on delivering their wish list. Table 7.7 lists the information supporters need from the other groups

At the end of the postdelivery period, during decommissioning, supporters will be involved to review the decommissioning plans and aid in rehearsal of the decommissioning processes. Supporters will need to assess what parts of the QA and QC library and other data are archived, and what is destroyed. This includes considerations of legal issues with compliance and with data protection.

7.6 Summary of the group

Supporters must communicate well with the other groups if they are to be effective. They have a vital role to play in the success of the organization, in supporting all the groups to get the best from IT. This group has a deep understanding both of the technical risks to systems as changes are introduced and of the real needs of the customer. Having worked with the

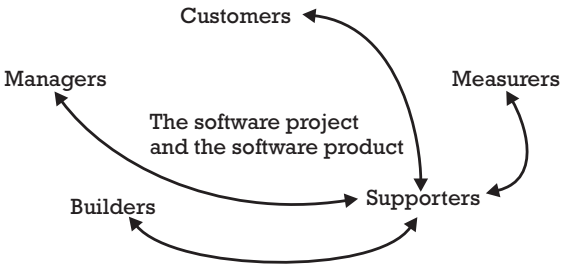


Figure 7.1 Communication between groups.

Table 7.6 Information That Supporters Have That Others Need

Before the SDLC starts and updated throughout the SDLC	Possible solutions to problems/ideas;
	Technical constraints and why these are constraints, for example, in the environment;
	Why this problem/idea (e.g., for a technical problem) is important and the impact on the customers for business as usual;
	Whether proposed aims/indicators are understood;
	Proposed objectives/targets for the solution;
	Whether proposed customer acceptance criteria are SMART (specific, measurable, achievable, realistic, and time-bound);
	Identification of supporter acceptance criteria;
	Technical risks (likelihood of this going wrong) and technical constraints;
	Precision and accuracy of estimates, when refined estimates will be possible;
	Constraints on SLAs postdelivery;
	Scope of supporters' operational acceptance testing.
During the SDLC	Manufacturing/product quality viewpoint—what attributes are of most importance and acceptance criteria for these attributes;
	User quality viewpoint—the operational, fit-for-purpose view of the software, with updated information as this changes over the SDLC;
	Technical impact of proposed changes in scope;
	Review and testing comments on proposed products to measure how closely they meet acceptance criteria, especially for nonfunctional attributes;
	Changes in IT infrastructure, systems, software, and other services that may affect the SDLC plans.
At delivery	Confirmation that the delivery is complete and acceptable.
Postdelivery	Evaluation of the software, how it affects the SLAs, whether it is fit for purpose, number of defects identified, improvements, and optimizations;
	Evaluation of the processes used and how they could be improved;
	Number of defects reported during live use;
	Customer satisfaction with the delivered products;
	Supporter satisfaction with the delivered products;
	Efficiency and effectiveness of quality processes;
	Improvement suggestions for all processes;
	Advice on impact of change.

customers' delivered software, they understand how it stands up to real use. They have knowledge of the risks to the business of software failure and which areas of the software are critical to the organization. They can and must be involved throughout the life span of the software, from conception and through the SDLC, as well as during delivery and use of the system until decommissioning.

7.7 Summary of all the groups

All the groups are important to the success of the IT provision for an organization. We have seen in Chapters 3 to 7 that each group has information

Table 7.7 Information That Supporters Need from Others

Managers	Organization goals and strategy; Changes to the goals and strategy, immediate tactics; Problems and issues; Forecast requirements for IT and infrastructure; Feedback on services provided; SLA requirements; Customer acceptance criteria.
Builders	Cost and resource constraints and why these are constraints; Value quality viewpoint; Whether proposed aims/indicators are understood; Proposed objectives/targets for the solution; Whether proposed acceptance criteria are SMART (specific, measurable, achievable, realistic, and time-bound); Nontechnical risks (likelihood of this going wrong) and nontechnical constraints such as resources, time, budget, and people availability; Precision and accuracy of estimates, when refined estimates will be possible; Feedback on services provided.
Builders	Manufacturing/product quality viewpoint; Whether proposed aims/indicators are understood; Proposed objectives/targets for the solution; Whether proposed acceptance criteria are SMART (specific, measurable, achievable, realistic, and time-bound); Technical risks (likelihood of this going wrong) and technical constraints; Precision and accuracy of estimates, when refined estimates will be possible; Update on progress; Environmental requirements for the SDLC; Environmental requirements for delivery and live use; Feedback on services provided.
Other supporters	Feedback on progress with problems, incidents, and issues; Changes in risks; Feedback on services provided; Supporter acceptance criteria.
Measurers	Manufacturing/product quality viewpoint; Whether proposed aims/indicators are understood; Proposed objectives/targets for the solution; Whether proposed acceptance criteria are SMART (specific, measurable, achievable, realistic, and time-bound); Technical risks (likelihood of this going wrong) and technical constraints; Precision and accuracy of estimates, when refined estimates will be possible; Update on progress; Environmental requirements for the SDLC; Environmental requirements for delivery and live use; Feedback on services provided; Advice on acceptance testing; Advice on review processes; Assurance that QA and QC activities have taken place; Results of those activities; Aid in using tools and techniques (e.g., regression test packs); Aid in using known problem lists and workarounds.

that the others need, and that this information may change throughout the SDLC and the life of the software during use.

Customers provide a user-based, fit-for-purpose view, which changes continuously to meet the changes in the real world. Managers understand the nontechnical constraints on the organization and act as an information conduit between the groups. Builders use their technical knowledge to provide solutions to the customers' problems. Measurers provide information that allow all groups to decide on whether the products are of an acceptable quality for use and optimization over time. Supporters provide an understanding of technical risks to business as usual, together with the supporting infrastructure for all the groups' activities.

In Chapters 8 to 12 we will follow the life of software from its conception to its decommissioning, and we will look at the communication methods and techniques we can use to help these five important but disparate groups understand each other and relate to each other's needs and strengths.

References

[1] IT Infrastructure Library, *Best Practice for Application Management,* London, England: Office of Government Commerce, 2002.

[2] International Standards Organization/International Electrotechnical Commission (ISO/IEC), DTR 9126, Software Engineering—Software Product Quality (Parts 1–4, 2000/2001).

[3] IT Service Management Forum, http://www.itsmf.com, October 2003.

[4] Quagliariello, P., "Introduction to IT Service Management, ITIL, and ITIL Capacity Management," http://www.pultorak.com/home/speaking_engagements/presentations/2003_03_07_cmg.pdf, October 2003.

[5] European Foundation for Quality Management, "EFQM Excellence Model" and "Fundamental Concepts of Excellence," http://www.efqm.org, August 2003.

[6] Malcolm Baldrige model, http://www.quality.nist.gov/index.html, August 2003.

[7] IT Infrastructure Library, *Best Practice for Service Support,* London, England: Office of Government Commerce, 2002.

[8] IT Infrastructure Library, *Best Practice for Service Delivery*, London, England: Office of Government Commerce, 2002.

[9] IT Infrastructure Library, *Best Practice for Security Management,* London, England: Office of Government Commerce, 1999.

[10] The Stationery Office Web site, http://www.tsonline.co.uk/bookshop/bookstore.asp?FO= 1150345, October 2003.

[11] IT Infrastructure Library, *Planning to Implement Service Management,* London, England: Office of Government Commerce, 2002.

[12] IT Service Management Forum, "BS 15000 IT, Service Management," http://www.bs15000certification.com, October 2003.

[13] British Computer Society Qualifications, http://www1.bcs.org.uk/link.asp?sectionID=574, September 2003.

[14] British Quality Foundation, "Recognition and Reward: Keep Your Staff Smiling," *UK Excellence*, August/September 2003 (whole issue devoted to rewards and recognition).

[15] Warden, R., and I. Nicholson, *The MIP Report—Volume 2—1996 Motivational Survey of IT Staff,* 2nd ed., Bredon, England: Software Futures Ltd., 1996.

[16] Beck, K., *Extreme Programming Explained—Embrace Change,* Reading, MA: Addison-Wesley, 2001.

[17] Gilb, T., papers on Evolutionary Delivery on http://www.gilb.com.

[18] The Office of Government Commerce, *How to Manage Business and IT Strategies,* London, England: HMSO, 2002.

[19] The Office of Government Commerce, *How to Manage Business Change,* London, England: HMSO, 2002.

Selected bibliography

IT Service Management Forum, *A Dictionary of IT Service Management: Terms, Acronyms and Abbreviations: Version I,* London, England: The Stationery Office, 2001.

IT Service Management Forum, *A Dictionary of IT Service Management: Terms, Acronyms and Abbreviations: Version I (North America),* London, England: The Stationery Office, 2001.

IT Service Management Forum, *IT Service Management: A Companion to the IT Infrastructure Library: Version 2,* London, England: The Stationery Office, 2001.

CHAPTER

8

Contents

The Life Span of a Software System

In this chapter I shall:

- Introduce the concept of software life span;
- Describe the stages in a software system's life span;
- Discuss where the quality definitions from Chapter 1 best fit in the stages;
- Introduce the concept of entry and exit criteria between stages;
- Prepare you for the detailed stage descriptions of Chapters 9 to 12.

As a business, we're crippled by our system! My development manager has just told me that removing the annual management charge from our premium account is going to take 50 days' effort. Our competitors do this sort of thing in 5 minutes. Unbelievable!

—Chief operating officer (COO), lamenting those of us who do not think beyond implementation

8.1 Life span or life cycle?

How do we avoid the problems upsetting the irate COO? To help, I am suggesting that we focus on the "life span" of a software system. The dictionary defines life span as "the length of time for which a person or animal lives or a thing functions" [1]. The life span of a *software system* begins when someone identifies a business problem that may be solved with software, continues through build and delivery, and, finally, ends with its decommissioning. The life of a software system includes what is often referred to as the software-development life cycle

(SDLC), which includes the design, build, testing, and delivery of the system. For most systems, the SDLC is a relatively short part of its life span.

Most systems spend the greatest part of their life span in use, being supported and maintained [2]. For example, many of you will remember the worries before January 1, 2000, that systems would not cope with years starting with "20" rather than "19." Some of the systems in question were more than 20 years old and still in use. Many organizations continue to depend on such software. Why replace something that works? Modern systems show the same pattern of a relatively short SDLC, followed by a long life of use and maintenance. Web sites spring to mind: After the initial launch they are continuously used and simultaneously changed.

Thinking about the life span of a software system instead of focusing on its development life cycle forces us to take a wider view of quality. We are more likely to consider the value view. This is surely beneficial, as many organizations struggle with the cost of ownership for their legacy systems. We will also think about the user view: how the customers will deploy the software. We may even identify the requirements of those tasked with supporting it. Understanding the life span of software affects our interpretation of the product and manufacturing views. We may develop tactical software intended to meet a finite need (short life span) very differently from a core system our organization intends to use for many years (long life span). In either case, the software is only there to support other activities. The McCartney report emphasizes that "Delivering IT is only ever part of the implementation of new, more effective ways of working.... Achieving this requires a clear vision of the context in which IT is being implemented" [3].

Because the SDLC is often loosely referred to as the software life cycle, I have chosen to use "life span" to refer to the entire life of a software system, in order to differentiate the initial development from the rest of the software's existence.[1]

The purpose of this chapter is to outline the four main stages I will distinguish throughout the life span of a software system. These four stages I have called *start-up, development* (the SDLC), *delivery,* and *postdelivery* (when the customers use the system until it is decommissioned). Maintenance changes are made to the system during postdelivery. You may use different names for these parts of the life span in your organization, or you may find that an SDLC is defined but not start-up, delivery, or postdelivery, or that all four stages are regarded as part of the SDLC. Do not worry about that; the naming and organization of the stages is simply to help us manage our work. The division into four life span stages that I am making is very simple (see Figure 8.1).

1. The IT Infrastructure Library refers to the "Application Management Lifecycle," containing Application Development (which includes the SDLC), and "Service Management" for the postdelivery activities. I have decided that for this book, *life span* and *life cycle* are simpler and easier to differentiate [4].

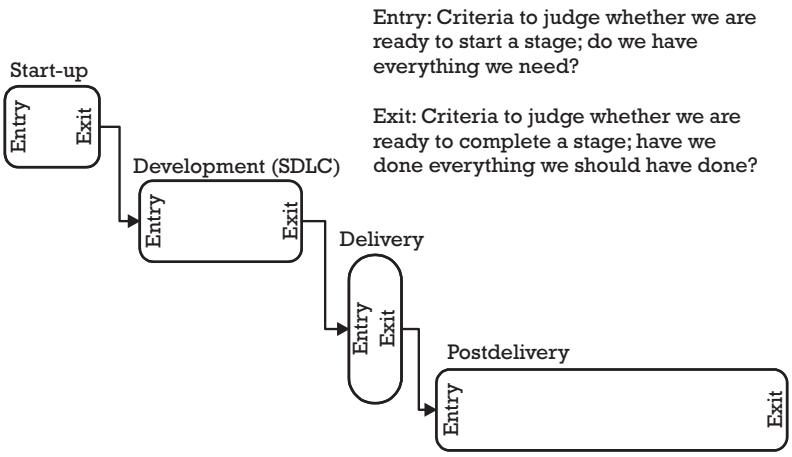


Figure 8.1 Life span stages.

8.1.1 Start-up

At this stage, members of the customer group realize that they have a problem that needs to be solved. They request the provision of a software solution. I use the word *problem* here; it may, of course, be an *opportunity*, for example, to expand into a new marketplace; or *an outside requirement*, for example, a legislative change, rather than a problem. In a healthy organization where relationships between groups are strong, managers and builders in IT may trigger start-up. IT can *create* opportunities by suggesting technical possibilities unknown to the customers. For example, in a major supermarket group, the customers knew that ordering stock for Christmas was a nightmare, and accepted that they would overorder and run out of products. Surely, that is part of their business, isn't it? The IT director identified a software package that could manage the inventory more effectively, based on predicted demand. The customers were unaware that this was possible, but were delighted when the project had an almost immediate payback.

In start-up, we must understand the problem, and if we do not, explore it more. We need to decide whether a software solution is appropriate, and, if it is, agree on the overall constraints, acceptance criteria, and thus the type of SDLC. If we decide to follow a software solution, we need some formal or informal contract between the groups. This is covered in Chapter 9.

8.1.2 Development

At this stage, a software project is undertaken, through the SDLC. This includes requirement definition, design, build and test activities to bring the software solution from the high-level definition of the start-up stage to a system ready for live delivery. It is important to remember that this is not just code being built by developers. Remember our poor trainer from Chapter 1. People will also be busy designing, building, and testing training, documentation, and support materials. If the software being developed is a package for commercial sale, the support material may include sales and

marketing material. For an in-house business system, it may include business process manuals. These are all part of the delivered product.

The work required might include the acquisition of a commercial off-the-shelf (COTS) system, building a custom-made system, tailoring a package, or making changes to an existing live system.

Members of all groups will be involved, directly or indirectly, but the focus is on the builders and measurers.

There are several different life-cycle models for software development. Some of these have been in use for many years, but new models and variations on the models have been developed over the years. The changes to the models have been made as customer problems, perceptions of quality, and the technologies available have evolved. I will look at the main groups of models. We will examine their advantages and disadvantages, and consider how to combine their best aspects to customize a life cycle. This is covered in Chapter 10.

8.1.3 Delivery

This is the point where the completed software solution is moved to live use. It is a time of anxiety for all the groups. The customers take delivery of the software and start to use it. Members of all groups will support the delivery, but their viewpoints will be quite different. The customer and supporters are very engaged at this point. Will the system really work? Will they be able to use it? Will the delivery go well? The builders and managers have quite a different reason for anxiety at this point. Will the project team be able to finish this project and (with more or less relief) move onto a shiny new project? For the measurers, fairly or unfairly, their efforts in assessing quality are now under the microscope. Did they make the right assessment? At delivery, many activities will come together at a single milestone in the project plan. This is a short, intense period of stress, anxiety, and emotions that are a rollercoaster between despair and elation. I will discuss delivery in Chapter 11.

8.1.4 Postdelivery

This is the life of the software while it is being used until it is decommissioned. It is the longest part of a software system's life span [5, 6], and is where the benefits of the software are realized; "A project or program is only successful if it delivers the benefits for which it was initiated" [3]. Customers use the software while supporters maintain and support it. Although the project team has often disbanded, all groups have an interest in quality at this stage (see Section 8.3). Additionally, it is almost certain that the software will undergo maintenance changes. Customers will identify new problems to be resolved and new opportunities to grasp, and will encounter new outside requirements. The system may need to be enhanced, corrected, and adapted to changes in its environment. To aid this, supporters may need to change it to make future support and maintenance easier. So the system will be subject to successive cycles of change, often with many changes taking

place in parallel (Figure 8.2). Each maintenance change will have a start-up (SU), development (SDLC/SMLC), and delivery (D), feeding into the post-delivery (PD) system. Figure 8.2 shows new SDLCs, involving the whole cast of characters from before, that could be described as SMLCs (software-maintenance life cycles). This is covered in Chapter 12.

8.2 Entry and exit criteria between stages

Each stage has within it some entry criteria (the criteria for starting that stage) and some exit criteria (the criteria for completing the stage). Some people call these the "quality gates." The entry criteria to a stage must match the exit criteria from the previous stage, like a series of jigsaw puzzle pieces. This may seem obvious, but I have seen projects affected badly by misunderstandings at the hand-over points between stages as confusion arises between people about responsibilities, authority, acceptance criteria, and quality. The reason for entry and exit criteria is to clarify to people how and when a handover can take place, and when it should *not* take place. In a project or program with several parallel activities, a set of exit criteria may feed more than one set of entry criteria, and a set of entry criteria may need exits from more than one previous activity. Long stages, for example the SDLC itself, will have entry and exit criteria to control the steps within the stage. We will see examples of this in later chapters.

Each entry and exit point needs an owner. This is someone who has the knowledge and authority to judge that the criteria are complete, by using a checklist [7] or by best judgment. The former is more objective; if the checklist is complete we move on, otherwise we do not. Using best judgment alone can result in an argument about whether the criteria are fulfilled or not, especially if the ownership and responsibility for the quality gates are not clear. Sometimes, we want to ignore the criteria. For example, during

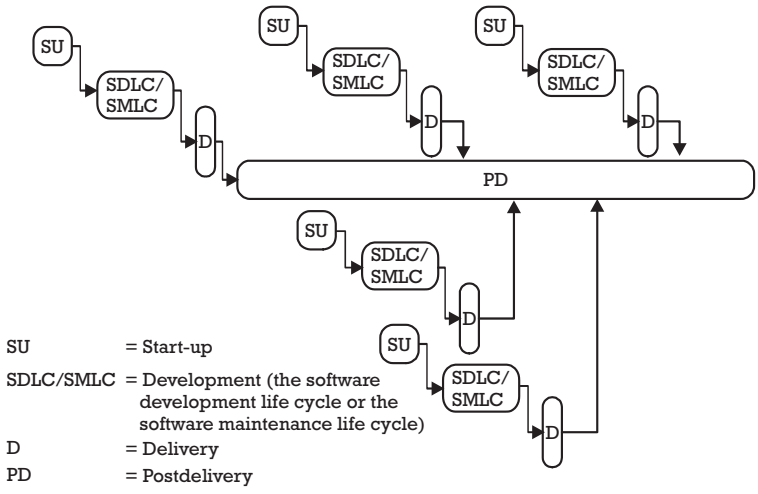


Figure 8.2 Change cycles to a software system.

start-up the project team may want to start work before all the acceptance criteria are set, in the belief that this will save time. If this is the case, we need to examine *why* we set the criteria in a particular way and the risks we take in not fulfilling the criteria. As we will see in Chapters 9 to 12, the precise entry and exit criteria depend on the situation, so checklists will need to be tailored. As well as an owner, identify an escalation authority, to make judgments in borderline cases.

Table 8.1 gives an example of simple entry and exit criteria between stages; we will discuss them in detail within each stage. Typically some type of review is required to check that the criteria have been fulfilled properly; for example, [3] suggests a peer review at "project gateways."

8.3 Changes in quality viewpoints across the life span of a system

Software quality is important throughout the life span of a software system. It can be built in or neglected at each stage of a system's life span. In Chapter 1, we looked at five definitions of quality and at associated quality viewpoints. Now I will consider how these viewpoints change across the life span of the software system.

The dominant group at each stage will dictate the prevailing view (see Table 8.2), but each group will still hold onto its preferred view, sometimes even feeling that their view is being discounted. For example, during development, the easiest view for the project team to use—the manufacturing view—will mean that they concentrate on delivering to specification.

Table 8.1 Example of Entry and Exit Criteria for Stages in the Life Span

	Entry	*Exit*
Start-up	Problem identified	Contract for SDLC
Development	Contact for SDLC	Acceptance test passed
Delivery	Acceptance test passed	Load to live use complete
Postdelivery	Load to live use complete	Decommissioning complete

Table 8.2 Prevailing Views of Quality Across the Life Span

	Stage			
Quality View	*Start-Up*	*SDLC*	*Delivery*	*Postdelivery*
Transcendent view	✓	✓	✓	✓
User view	✓	(✓late)	✓	✓
Value view	✓			✓
Product view		✓		✓
Manufacturing view		✓		✓

✓ is a primary quality view.
(✓) is a quality view that may be taken by some people in this stage.

However, when delivery happens, the user view will prevail. The builders feel pleased that they have delivered to specification, and are then stunned to find the system rejected by the customers.

This happens more often than one might suppose, either because the specification was wrong, or because it was misinterpreted, or because the world has changed so that the specification is out of date by the time the system is delivered.

Once the system is in use, we can measure its effect. The managers will be interested in measures of value for money and return on investment, supporting the value-based view of quality. Supporters, measurers, and builders will be interested in measures of defects reported during use, supporting manufacturing- and product-based views of quality. Customers and supporters engage with the software day to day. They hold a transcendent view of quality ("Are we enjoying this system?") but also the user-based view. They will have a perception of the software as fit for purpose or otherwise.

In the next four chapters, I will describe the quality activities at each stage, showing what people can do as a team at each stage to build in and measure quality, therefore achieving software quality through teamwork.

References

[1] *Compact Oxford English Dictionary,* 2nd ed., Oxford, England: Oxford University Press, 2002.

[2] Hetzel, W., *The Complete Guide to Software Testing,* Wellesley, MA: QED, 1984.

[3] Cabinet Office, *Successful IT: Modernising Government in Action*, London, England: HMSO, 2000.

[4] IT Infrastructure Library, *Best Practice for Application Management,* London, England: Office of Government Commerce, 2002.

[5] Baxter, I. D., and C. W. Pidgeon, "Software Change Through Design Maintenance," *International Conference on Software Maintenance* (*ICSM '97*), 1997.

[6] Yourdon, E., "Long-Term Thinking," *Computerworld,* October 2000, http://www.computerworld.com/managementtopics/management/story/0,10801,52398,00.html, accessed January 2004.

[7] Gilb, T., and D. Graham, *Software Inspection*, Reading, MA: Addison-Wesley, 1993.

CHAPTER

9

Contents

Start-Up for a Software-Development Project

In this chapter I shall:

- Describe the entry criteria, steps, and exit criteria for the start-up stage;
- Show how to use this stage to decide whether software development is required;
- Identify the quality definitions from Chapter 1 that prevail at start-up;
- Identify how each of the groups identified in Chapter 2 becomes involved at start-up;
- Introduce techniques for analyzing problems and solutions, setting aims and objectives, and defining acceptance criteria and constraints.

Is that what you thought the reason is for this project? That's not why I want it done....

—The development and business managers discover, several months into the project, that they have opposing goals

I would have thought it was obvious that we wouldn't want this!

—Customer rejecting software on delivery

9.1 Start-up—description

We realize that we have a problem, or an opportunity (for example, to expand into a new marketplace), or an outside requirement such as a legislative change. We must understand it and decide how to resolve it before we launch into delivering a solution. Although many of us see quality as delivering a project excellently, the most important project decision is

ensuring that we do the *right* project. If the builders do not understand the problem and the customers' and managers' quality views and their constraints, they may build software that is too costly or not fit for purpose. The cost of making that mistake is huge compared with setting up the project correctly in the first place: One commentator reported that the cost of failed IT projects in the United States was estimated at $84 billion [1]. The cost of finding our mistakes and sorting them out escalates throughout the project [2]. The cost of doing the *wrong project* could be phenomenal; our biggest risk is spending a lot of money and *not* resolving our problem. However, once we identify that we have a problem, our impulse is to start doing something, *anything*, to show that we are taking action. This means that we dive straight into delivery without really knowing what we want to do. Our projects go wrong before they have started! So, in this chapter, I am going to show you how to make the difficult move from a complex and messy business problem to a project that is clear and structured. We will set the foundation for future work, possibly including a software-development life cycle (SDLC) and delivery (see Figure 9.1).

As an SDLC is costly, it is worth spending some time and effort doing this stage well. We might decide that we cannot resolve this problem by a software solution, or that we prefer to use a manual solution, or to do nothing. We must:

- Explore the problem until we understand it.
- Decide whether this problem is worth solving.
- Set general constraints and parameters for the solution.
- Decide how to approach the problem; for example, a software solution may be appropriate.
- Agree on a formal or informal contract of work, with constraints, acceptance criteria, and an outline or high-level plan.

As we see in Figure 9.2, we will not just work our way down the list, checking each off in order; we will find that we need to revisit interrelated tasks as we investigate the problem. We cycle round, investigating the problem until we have enough information to either decide to do nothing, to have a manual solution, or to have a software solution to the problem.

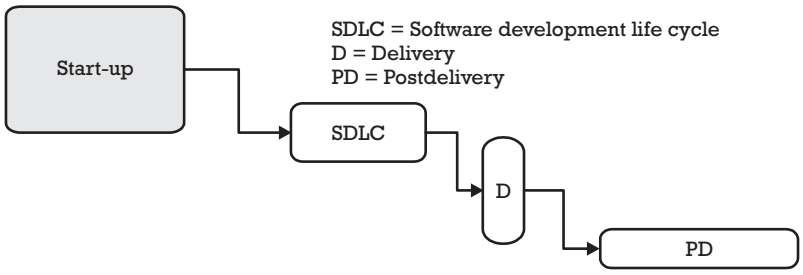


Figure 9.1 Life-cycle stage diagram.

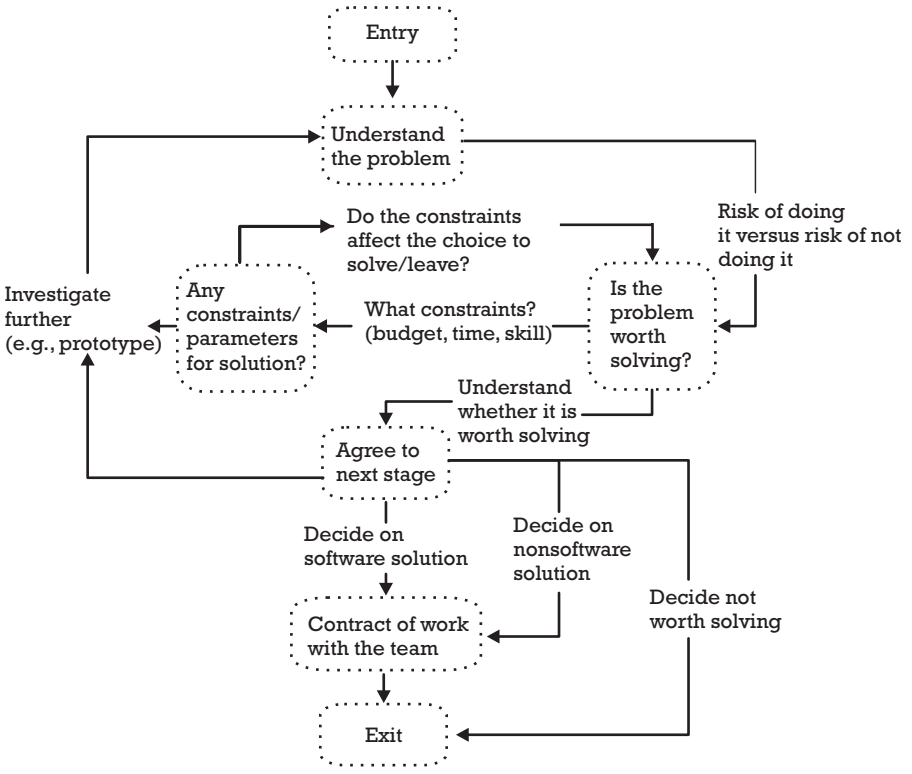


Figure 9.2 Task summary for start-up.

9.2 Start-up viewpoints

A factor I often see contributing to the failure of software projects is that people with useful viewpoints and information are not involved early enough. Often, the only groups that participate in start-up are the customers and managers. Table 9.1 shows the quality views represented.

The team has a biased quality viewpoint, as it lacks either the product or the manufacturing view. Also, because they are not involved, the builders, measurers, and supporters are not being directly exposed to the *value* and *user quality views*. The consequences of this can be severe. In a large transformation program in financial services, a group of consultants found that the IT teams had little understanding of the business cases for their projects. They did not know why they were building, so were unaware of the

Table 9.1 Views of Quality at Start-Up: Simple View

	Groups	
Quality view	*Customer*	*Manager*
Transcendent view	✓	✓
User view	✓	
Value view		✓
Product view		
Manufacturing view		

organizational impact of going over budget or missing scope. They delivered software that often failed to meet business expectations.

Table 9.2 shows that involving all the groups covers the quality viewpoints and thus provides wider knowledge of the problem and potential solutions.

Some IT people, including builders, measurers, and managers, express frustration that the customers often do not know what they want, but we must realize that it *is* difficult to understand many problems. Often, when customers call in consultants "All [they know] is that something is not working right and [they] need some kind of help" [3]. In some situations, there is no clear customer group. Suppose your organization builds software packages that are purchased by customers "off the shelf." You may not have direct contact with your "real" customers. Sales, marketing, or market research people could represent the customers or involve customers via market research. In one company producing project management packages, the "key customers" for each release were representatives from group marketing and the customer help desk.

9.3 Entry criteria for start-up

The entry criteria for a stage are the rules or reasons by which we decide we are ready to start that stage. This might include a list of input deliverables or a list of events that have to have happened. For the start-up stage, these are quite simple (Table 9.3); we realize we need something—a problem solved, a change in our processes, a new product supported. We have an idea for an improvement or a change. This may be enough, but in some organizations it

Table 9.2 Views of Quality at Start-Up: Teamwork View

	Groups				
Quality view	*Customer*	*Manager*	*Builder*	*Measurer*	*Supporter*
Transcendent view	✓	✓	✓	✓	✓
User view	✓			(✓)	✓
Value view		✓			
Product view			✓	✓	✓
Manufacturing view			✓	✓	

✓ is a primary quality view.
(✓) is a quality view that may be taken by some people in this group.

Table 9.3 Example of Simple Entry Criteria and Owners for Start-up Stage

Example Entry Criterion	*Example Owner*
System or process problem identified	Change control group
Legislation change identified	Audit and compliance
Idea for new product identified	Marketing
Process improvement identified	Process owner

may be the entry into a suggestion scheme or a change process. When you define entry criteria and steps for your start-up stage, look at your existing scheme, and decide what ideas from this chapter you would like to add to improve it.

9.4 Start-up—typical activities

9.4.1 Understanding the problem/idea

Once a problem or idea has been identified, we can use various methods to analyze it. Remember that I am using "problem" as shorthand for "problem, outside requirement, opportunity, or other reason for wanting to change the status quo." This step is critical, so beware underplaying it. If you get this bit wrong, the project could *already* be doomed to fail! People will have preconceptions about what needs to be done, even if they are using the word "problem," as TQMI[1] [4] identifies:

- *Solution*: "our problem is that we need a new Web-based interface."
- *Symptom*: "our problem is people keep making mistakes in data input."
- *Decision*: "our problem is deciding which supplier to choose."
- *Problem*: "our problem is that it isn't working as we expected."

Involving all groups will help prevent you from confusing the symptom with the cause. Be radical and dig deep until you find the real problem. You will need to consider many possible and underlying causes; for example, a customer complaint might have one or more root causes (Table 9.4).

There are many methods for understanding problems and solutions. Some you will find particularly useful are cause–effect or root-cause analysis, solution analysis, brainstorming, and prototyping (see Table 9.5). You will probably use cause–effect/solution (C-E/S) analysis and brainstorming on your first pass, followed by methods such as prototyping as you start trying to identify possible solutions (see Figure 9.3).

Table 9.4 Severe Customer Complaint Root Causes

Problem	*Severe Customer Complaint*
Apparent cause	Poor quality products
Possible underlying causes	Product is badly designed/badly built/fails too quickly, and too often.
	Product not matched to customer need/misadvertised/missold.
	Customer expectations not managed against feasible targets/too expensive, late.
	Poor quality service/customer complaints not handled well/customer passed from person to person, multiple calls required to resolve problem.
Root cause in this case	The customer had a minor problem that escalated as multiple calls were required to resolve it.
Solution	Improve customer complaints procedure to ensure a fast resolution to problems.

1. Reproduced by permission of TQMI Ltd., from *Problem Solving—Tools and Techniques.*

Table 9.5 Summary of Techniques for Start-Up

Subject Area	*Technique Examples*	*In 30 Seconds… (see Appendix A for more)*
Identify problems and root causes, find solutions	Ishikawa fishbones [4, 5]	Use to identify problems, root causes of problems and solutions. On a fishbone diagram, brainstorm problems, their possible causes, their root causes, and, therefore, solutions to the root cause. See Chapter 12 for an example.
Review documents and other products	Walkthrough [6], inspections [2, 6], peer review [6]	Walkthrough is a review with the purpose of increasing understanding of a document. The author introduces the audience to the document and takes them through it, explaining the content. Inspection is a formal review with the purpose of identifying and preventing defects and nonconformities to standards and specifications. Peer review additionally allows discussion.
Improve meetings and improve contributions	De Bono's Six Thinking Hats [7]	Improve meetings by setting rules for behavior. Six "hats" are used. Everyone wears the same color hat at the same time: Blue Hat—meeting structure, Black Hat—pessimistic, Yellow Hat— optimistic, Red Hat—feelings, White Hat—facts, Green Hat—creative ideas. Allows meeting members to move outside their stereotypes and allows time for different, sometimes difficult types of communication.
Helping groups agree on aims, objectives, targets, and indicators	Weaver triangle [8]	On a one-page diagram, the group identifies and agrees on the aim of the project (why it is being done) and associated indicators of success, then on the objectives of the project (what is to be done) and associated targets. They identify where stakeholders have different aims for the project.
Understand/ explore ideas	Prototyping [9, 10]	Not an SDLC! We prototype to try out ideas or, if we are not sure what we want, we build a model of a possible solution(s). Discuss the prototype and review it. Two types of prototyping in software are "lo-fi" and "hi-fi." In hi-fi (high-fidelity), we build screens like those the customer will see. Lo-fi (low-fidelity) prototyping uses paper/ white board. In lo-fi there is no danger of believing the prototype is the software.
	Modeling and picturing, stories, metaphors [11–13]	There are a number of modeling and picturing techniques that can be worth exploring with lo-fi prototyping. These include Rich Picturing and Mind Mapping. Sketching a picture of a problem, solution, or idea can clarify it. Use stories, metaphors, and analogy as well; they can help understanding.
Understand whether an idea is worth pursuing	Risk workshop by brainstorming [4, 5]	A brainstorming workshop is run to list all the possible risks people can identify. Risks are sorted into groups, also separate risks (might happen in the future), issues (problems right now) and constraints. Risks are scored for impact and likelihood, and ranked to give a prioritized list.
	Cost–benefit analysis [5]	Cost–benefit analysis is done by calculating the predicted benefits of the proposed change (time, money saved) and setting this against the predicted cost.
Choose which idea to pursue	Pareto analysis [4]	The Pareto principle suggests that 80% of the problems are the result of 20% of the causes. Gather data on the frequency of problems/causes, and then resolve the most frequent.
Team relationships and natural roles/team skills	Belbin team scores [14]	The SDLC can fail because of personal rather than technical factors. Teams need to understand their strengths and weaknesses as a team. A balance of roles/skills is required in the personalities in the team. Example roles: plants have new ideas; completer–finishers want to finish to fine detail. Too many plants and you will never finish anything.

Table 9.5 (Continued)

Improve communication—Empathy with others	MBTI [15], Honey & Mumford [16], Kirton [17]	Different people have different personalities and communication styles. People who wave their arms around and talk a lot can annoy people who like to be quiet and think, and vice versa. The Myers-Briggs Type Indicator (MBTI) identifies four contrasting type pairs, e.g., *Introvert/Extrovert*, leading to 16 "types" (e.g. INTJ is *Introvert-iNtuitive-Thinking-Judging*). The Honey & Mumford Learning Styles Questionnaire identifies preferred learning styles (e.g., *Pragmatists vs. Theorists* require different experiences to learn). Kirton identifies preferred problem-solving methods (*Adaptors vs. Innovators*)—do we break the rules or work within them?

If you do not have direct access to your customers, your sales and marketing group may use market research or ethnographical techniques at this stage. This includes observing or interviewing groups of likely or existing customers to help focus understanding of the customer's viewpoint, with use of prototypes. For example, Philips uses their "HomeLab" in Eindhoven to observe how people react to new technology, because it is only by observing people living with the technology that they can identify which ideas are good in practice, as well as in theory [18].

The output from this stage is a problem statement. This does not need to be formal. The aim is to provide a short, clear description of your analyzed problem. I would advise that you get others to have a peer review or a walk-through of the problem statement before you go any further into start-up.

Do not forget that allowing different viewpoints to be heard during discussion of ideas is important. People who are disillusioned may respond negatively to any new idea, including use of new techniques. If people say, "Yes, but…" or "We've tried that and it didn't work…," do not dismiss them. You will need to persist and show people that their contributions are

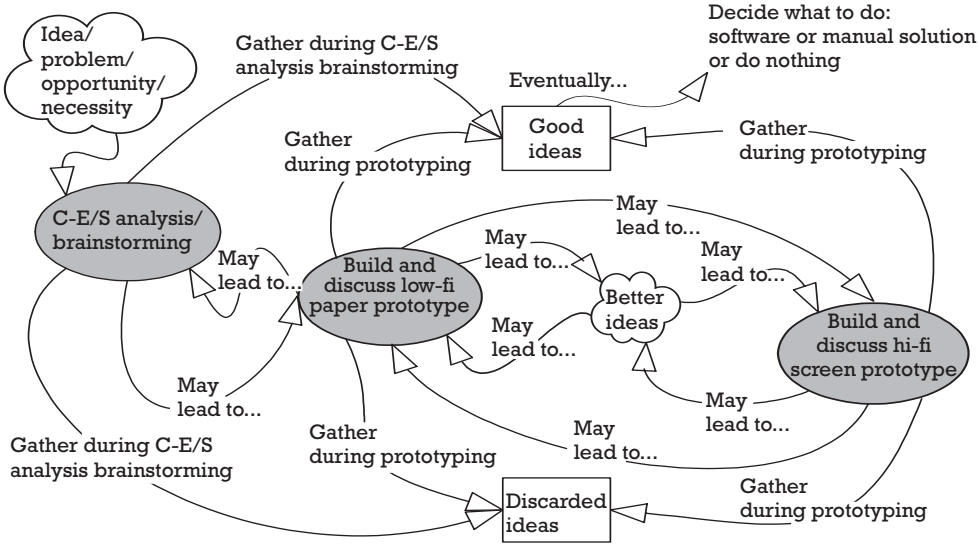


Figure 9.3 Problem and solution analysis cycles during start-up.

listened to seriously. People get into a habit of always reacting in the same way; they consistently either support or condemn suggestions [3]. Using De Bono's Six Hats [7] helps resolve this by encouraging everyone to express positive and negative views (see Table 9.5 and Appendix A). You may also find, as I have, that when you bring the groups together some people are very loud and others reticent. In certain organizational cultures, staff are reluctant to challenge or disagree with their superiors [3]. If you find this, then use techniques to help people express their views. For example, all ideas or comments could be submitted anonymously in advance of the workshop, then discussed "blind" (Frank Johnstone, personal correspondence, April 24, 2003). Technology can help; some organizations use online meetings and chat rooms to allow challenges to be presented safely.

Your view of the problem becomes more complete as you look at it from different quality perspectives. Every group should participate in workshops to identify problems and ideas, suggest solutions, build prototypes, and provide examples of similar business situations and how they were solved (Table 9.6).

9.4.2 Decide whether the problem/idea is worth solving

We now have some understanding of our problem or idea. We need to consider whether it is worth doing anything about it, and answering these questions will help us:

- What benefits will we realize by solving this problem?
- What risks are we taking if we do not do this?
- What risks are we taking if we do it and it goes wrong?
- What benefits might we *not* realize if we attempt it but it goes wrong?

Both action and inaction hold risks. For some of the entry criteria examples in Table 9.3, there is no doubt that a software project will be

Table 9.6 Group Contributions to Understanding the Problem

Activity:	*Group* *C = Customer, M = Manager,* *B = Builder, Me = Measurer, S = Supporter*			
Identify problem, idea	All groups			
Provide example of similar problems and solutions	All groups			
Take part in discussions and workshops	All groups			
Identify solutions and build prototypes	All groups			
Review role:	C	M	B	Me	S
Review/test ideas and prototypes against a quality viewpoint: U = User, V = Value, M = Manufacturing, P = Product	U V	V	M P	M P U	U P

undertaken. If a legislative change is required that affects the software systems, then the question is not "is this a good idea?" but "how shall we change the software?" If there is a choice and there may be risks in making changes, it might be better to accept the problem or not implement the new idea, having assessed the risk. For example, not implementing a new product gives us a risk of losing market share because our customers may move to our competitors. Implementing the new product may increase our number of customers and sales, and we should be able to predict the benefit gained. But supposing our infrastructure (warehousing, sales support, and so on) cannot support the increase? We may end up giving poor service to all our customers, and so lose more business than if we had done nothing.

We also need to consider whether there are risks involved with any expectations, for example the target date for getting the new product to market. What is the risk if we miss that date? What is the risk that the date will be too difficult a target for us to meet? You can see that there are two parts to the risk: we need to consider the likelihood that something will go wrong and the impact if it does go wrong. I have generally found that the customers, managers, and supporters have a good understanding of the impact of problems on the existing organization and infrastructure. Builders, measurers, and supporters understand where technical problems are likely to arise; they will understand the technical difficulties in solving the problem and the effect of constraints. Customers and managers understand where business complexity and constraints make problems likely.

The EFQM Excellence Model (see Chapter 1 and [19]) helps structure the way we think about the impact of either not doing something or doing it wrong. Consider the following:

- Impact on our organization's customers—service levels, goods, expectations, continuing delivery;
- Impact on the people in our organization—motivation, staff turnover, attitudes, development, loss of IT infrastructure/support;
- Impact on our partners and resources—support of systems and processes, damage to existing IT and other infrastructure;
- Impact on society—PR, attitudes, legislation, compliance;
- Impact on key performance results—financial, market share.

When we consider the *likelihood* of something going wrong if we *do not* take action and if we *do* take action, we look at our areas of weakness:

- Technical constraints and difficulties, for example, complexity, lack of knowledge of the technical aspects, limits to existing systems;
- Business constraints and difficulties, for example, complexity, lack of business knowledge, product or service delivery constraints;
- Use of, lack of, or misuse of tools, techniques, processes, and standards;
- Constraints, for example, in time, budget, personnel, skill levels, or other resources;

- Measurement of past experience and predictions from that.

The methods you can use were listed in Table 9.5 and are explained in Appendix A. They include risk workshops and cost–benefit analysis. You will probably need to consider risks and benefits along with constraints (see Section 9.4.3) because they will affect each other; if you only have the budget to build a family saloon car, do not expect it to be armor-plated. If you are deciding which of a group of problems to solve to get the greatest cost–benefit ratio, try using Pareto analysis (80% of the problems are the result of 20% of the causes), choosing the most frequent problems/causes to resolve [4].

9.4.3 Set general constraints and parameters for the solution

Some organizations like to keep the risk workshops, constraints definition, and cost–benefit analysis quite separate, but I have found that the constraints will often come out naturally during the risk workshop. You may want use part of the risk workshop specifically to collect constraints. The constraints, risks, and benefits tend to interrelate. If you have a constraint on timescale or budget, you may not be able to do as much as you wanted to do. This might increase your risks, but the risks associated with increased spending may be greater. Someone who took on a course of mine years ago gave me a wonderful analogy—it is like going to buy a washing machine:

> You go into the shop, and ask the sales assistant the price of washing machines. Suppose the assistant says that they range from $800 to $1,500. You only have $15. The assistant will point out that for that money, you can buy a bucket, a washboard, and some soap. You have kept within your budget constraints and avoided the risks associated with getting into debt. However, you are taking some additional risks—the washing will be done at a lower temperature, so it may be not so effective, and you are going to get soap on your skin (risk of "washerwoman's hand").

All the groups will be able to contribute to the discussion of constraints. Customers and managers will know about service level, timescale, budget, and resource and personnel constraints. Builders, measurers, and supporters will know about technical, knowledge, infrastructure, resources, and skills constraints. In particular, the supporters will understand the constraints against current IT service delivery and capacity [20, 21]. While examining the constraints, build prototype plans. Are any of the ideas "runners" within the constraints?

9.4.4 Agree on next stage

We have identified a problem or idea and we have looked at whether it is worth dealing with. Now we need to decide what to do. All the groups have

a contribution to make, so consider having a walkthrough of the information gathered so far, encouraging questions, comments, and additions. In a walkthrough, the author of a document takes the meeting through the document step by step, allowing them to ask questions and gather information (see Table 9.5 and Appendix A). This promotes greater understanding and helps find areas of ambiguity and mistakes. Display all the information gathered so that the whole group can see it. I prefer putting everything on the wall to allow everyone to look at it at the same time, standing and walking from chart to chart. It helps the team to discuss and think as a team. Use a data projector and a PC, or flip charts from the workshops. If people are geographically remote, consider on-line review and meeting tools, or videoconferencing.

Once the team has discussed the information, they can make a decision about what to do next. We have four options:

- Do nothing—decide that the problem is not worth solving. Log the decision.
- Do more work on investigating the problem. Perhaps do some prototyping, but check that you have the time, budget, and permission to carry on.
- Decide not to develop software, but to address the problem through your people or processes. For example, the solution might include recruitment, process change, or building a manual solution. Draw up a contract for the work.
- Decide that we need a software solution. Draw up a contract for the work.

9.4.5 Contract for work

Watts Humphrey remarked that one problem with software schedules is that managers view them as contract-like commitments but the software engineers (builders) do not view them as personal commitments: "Too often, software commitments are based on little more than hope" [22]. We need an agreement or contract between the groups. I am not (necessarily) talking about the legal contract between a customer and third-party supplier; it could be a formal or an informal contract between groups within an organization, but it is a commitment. The Personal Software Process (PSP) [22] and the Team Software Process (TSP) [23] emphasize the importance of this commitment, but how do we reach it? It is vital that everyone agrees on:

- The aims and the objectives for the work;
- The constraints for the work—dates, budget, people, resources, technical, and business;
- Acceptance criteria that define how we will know the work has been completed satisfactorily.

Everyone needs to agree on what has been planned and check that it is possible, by contributing to and reviewing the contract. This contract is the basis for the project(s) that will deliver the solution required by the customers, perhaps including an SDLC. Sometimes, it is easy to agree the contract; the problem or idea is well understood. Often, teams find this step difficult; perhaps the customers are not sure what they want, or cannot define it, or sometimes the builders and measurers are not sure they can build and test the software within the constraints. If the team cannot agree on the aims and objectives for the work, plus at least overall constraints and high-level acceptance criteria, it is probably not wise to agree on a contract for an SDLC; the problem or idea is not well enough understood to make a sensible commitment to the work.

There may be reluctance to contribute difficult or controversial ideas in contract negotiation meetings—"If I put my head above the parapet will I get shot?" This is particularly true if different groups disagree about what is *possible* within constraints. To prevent the loudest voice being the only one heard, use the techniques mentioned earlier to control contributions to the discussion.

9.4.5.1 Aims and objectives for the work

A project's *aims* describe why it is being undertaken. They answer the question, "Why are we doing this?" The answer is not "to build a Web site" but "to increase market share." Objectives are what you will do in order to reach an aim: "Our aim is to increase market share, so one of our objectives is to build a Web site." Each aim is associated with one or more objectives, otherwise it will not happen. Every objective must meet one or more aims, otherwise, why do it? Indicators measure the impact that the project has on the business. Did the project make a difference? Were the aims met? Has it realized benefits or mitigated risks? Indicators enable us to assess whether the aims have been met. Targets measure project delivery, monitoring whether the objectives have been achieved. We know whether a requirement is within our scope by testing to see if it aligns with the project aims and objectives.

A useful technique to help define and agree on aims and objectives is a Weaver triangle [8] (see Figure 9.4, Table 9.5, and Appendix A) as it provides a picture of the project on one page. I have used a Weaver triangle with software projects to demonstrate that key people around the project had radically different ideas of the real aim of the project. The reason one project was failing was that there was no agreement about why it was being done. Each team had taken off with different aims, in a different direction.

All the groups participate in defining and agreeing on the aims, objectives, targets, and indicators, remembering to check that they are SMART (specific, measurable, achievable, realistic and time-bound). If the team cannot agree on SMART aims and objectives, do not set the contract. Investigate the problem further.

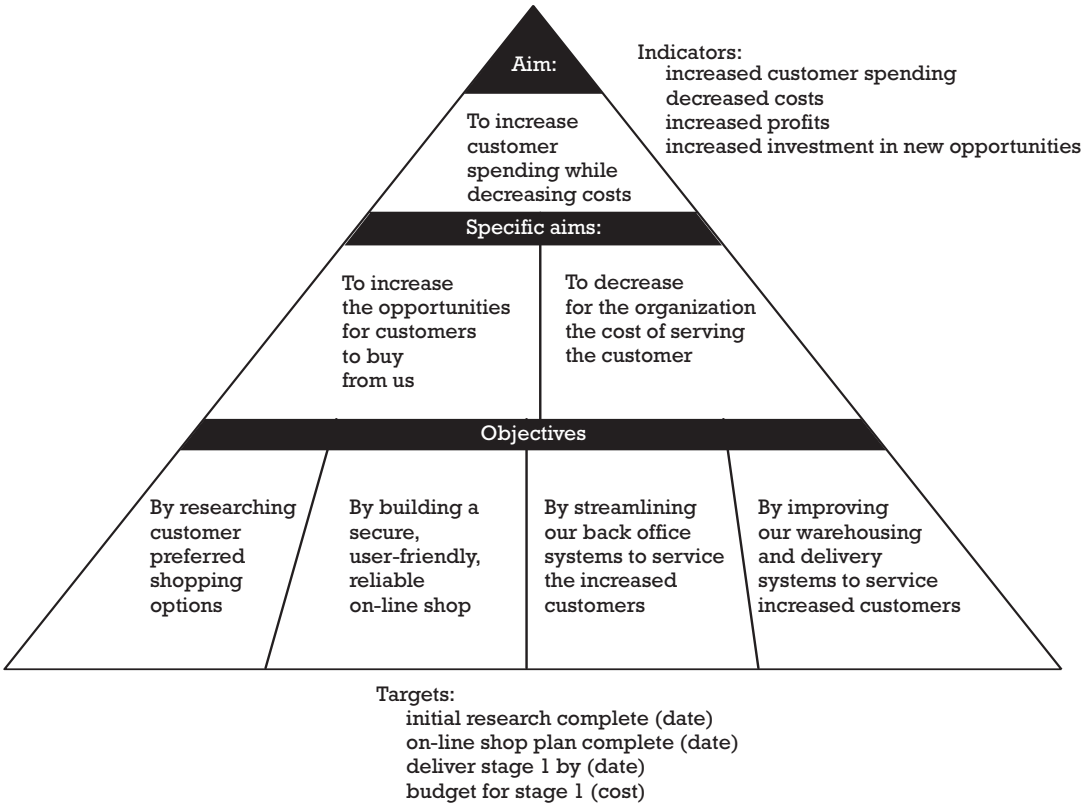


Figure 9.4 Weaver triangle. (*After:* [8].)

9.4.5.2 Constraints for the work

We have already looked at gathering the constraints. At this stage, we are confirming that they are correct and complete, and that the project can be delivered inside its constraints. In particular, managers will set date and budget constraints, against value measures. These need to be checked and agreed on by the other groups. As before, the builders, measurers, and supporters identify technical constraints, for example, IT service levels and capacity [22, 23]. These need to be understood and agreed on by the other groups. If the team cannot identify the constraints, at least at a high level, for example, the maximum budget, do not set the contract. Investigate the problem further and prototype the plan to see if it is possible to deliver within reasonable constraints. If there is some leeway in constraints, reflect this in the contract:

- "This date must be met—legal requirement" is mandatory.
- "Marketing have planned a July launch to meet the build up for the holiday market (November/December), latest launch is August" is mandatory with some leeway.
- "Would like to complete work before March next year" is a preferred date and could be renegotiated.

9.4.5.3 Acceptance criteria

Setting acceptance criteria at this stage is very important. The acceptance criteria are the means by which we know if a product or service is acceptable or not. They describe the attributes of a product or service, and the "pass mark" for each attribute for acceptability. If acceptance criteria are not defined or they are vague, we are in danger of building the wrong system. Like the targets and indicators, acceptance criteria must be SMART. Many projects are reasonably good at defining what the software should do (its *functionality*). Very few customers and managers, in my experience, give much thought to *how* the software should function. This is best illustrated by an example. An ATM machine should provide cash—that is its function. *How quickly* should it provide the cash—within 2 hours or within 1 minute? This is a *nonfunctional attribute*—it describes *how* the software works. They are often taken for granted. In the ATM example, it seems obvious that 2 hours is too long to wait for the money. The problem is that people's assumptions about these attributes may differ, so that although one might assume that the nonfunctional attributes are too obvious to mention, they may be interpreted differently by other people in the team. Whether the project includes an SDLC or not, acceptance criteria will be needed, in order that we can check that the project has delivered acceptable products and services.

Software standard ISO 9126 [24] describes attributes of software, and measures for those attributes, but, as you will see, we could adapt some of the attributes for nonsoftware solutions. If we use ISO 9126 to help us define acceptance criteria, we can improve their SMARTness. This increases the likelihood of the right software being delivered.

ISO 9126 breaks down the functional and nonfunctional attributes of software into a series of subattributes, questions and metrics. Some examples are shown in Table 9.7.

Let us take usability as an example. People find it hard to define and measure usability, so "give up" on trying to write acceptance criteria or trying to measure whether the software is acceptable. Look at Table 9.8, and you will see increasing refinement of one aspect of usability until we arrive at measurable acceptance criteria. Statements 1 and 2 are impossible to measure. No attempt is made to define "usability." Statement 3 crudely qualifies what usability means but it will be hard to measure, and it will, therefore, be difficult to design and build software that meets the customers' usability needs. Statement 4 is a little better; we know that some training will be needed but we have no definition of a new user. Is it someone new

Table 9.7 Examples of Attributes, Based on ISO 9126

Example Attribute	*Example Subattribute*	*Example Question or Statement*
Functionality	Suitability	Does the function perform tasks that do not conform to specified ones in requirements specifications or user manuals?
Usability	Understandability	Percentage of functions evident to user
Security	Access controllability	Is there any failing to defend against illegal access or illegal operation?

Table 9.8 Making Acceptance Criteria Measurable

1.	I assumed you would make it easy to use.
2.	It must be user-friendly.
3.	I want it to be as easy as the current system.
4.	A new user must be able to use it with 30 minutes training.
5.	Based on a sample of 20 typical users, at least 90% must learn to use the system in less than 30 minutes. After training, 100% of the sample users must be able to complete the standard "10 typical tasks" sequence without help, 95% of them completing it without help in less than 5 minutes.

to this particular software or to software in general? Statement 5 (based on ISO 9126 metrics under "Usability" for Learnability, Efficiency and Effectiveness) is measurable. Having more measurable acceptance criteria means it is easier to assess whether they meet other aspects of SMARTness.

The various nonfunctional attributes of software are of interest to different groups, so I would expect all the groups to be involved in setting acceptance criteria. Table 9.9 (based on discussions in 2002–2003 between David Hayman and me) lists attributes and groups. Remember when looking at the table that the customer group includes executives and sponsors as well as the software users.

Table 9.9 Selection of Nonfunctional Attributes and Groups

Attribute	*Group Setting Acceptance Criteria and Reason*
Performance (e.g., performance, stress, volume, scalability)	Customers—want good response from system
	Supporters—do not want bottlenecks in the infrastructure and systems
Security	Customers—Government and compliance requirements, CIA model (confidentiality, integrity, and accessibility of information)
	Supporters—infrastructure and firewalls
Reliability (e.g., reliability, availability, recovery)	Customers—service levels, product availability, lost sales, compliance, safety, avoid wasted time and lost work
	Supporters—reduce system restarts and help-desk calls, avoid wasted time and lost work
Usability	Customers—attractive system to use and to market, supports users at their skill level—not too easy, not too difficult—efficiency and effectiveness for users completing tasks, avoid wasted time, mistakes, increase productivity
	Supporters—reduce help-desk calls
IT support (e.g., installability, portability, compatibility, conversion, memory management, maintainability)	Supporters—need to support it within the existing infrastructure (e.g., amount of memory required, ease/cost of making changes)
	Managers—cost of changes to infrastructure, cost of making changes
Attribute	Group focus for reviewing acceptance criteria
All attributes reviewed	Measurers: they are testable (specific and measurable)
	Builders and supporters: they can be delivered within the constraints (achievable and realistic)
	Customers: they are what is required (realistic and time-bound)
	Managers: they are within cost/time constraints (realistic and time-bound)

Some of these attributes are specific to software, but others could be applied or adapted to nonsoftware projects. For example, suppose our project is to write some manual business processes. We would still look at attributes and acceptance criteria, for example:

- Usability—people are able to learn and use the new processes to carry out their tasks;
- Maintainability—able to update the business processes;
- Performance—throughput of tasks and business processes bottlenecks.

Just as in Chapter 1, where we saw that there is no "right" definition of quality, there is no "right" set of acceptance criteria. It is situational. Contrast the following:

- Air traffic control system—safety critical, so the emphasis is on reliability, recovery, performance, and security, but usability will also be important.
- Package to improve usability of Web pages for the visually impaired—it is browser-based, so usability and maintainability will be important.
- Software to launch an innovative new product and achieve "first-mover" advantage—marketing may produce a high initial demand and a new user group so stress and usability will be important.

We might believe that in a perfect world we would have no constraints, but this is not reality. For most commercial organizations, it would be unnecessarily expensive to have all the attributes "at 100%." Usually, the customer is best served by balancing the quality viewpoints. This means *planning* to deliver some attributes at a lower standard than what would be achievable in a perfect model. This is necessary to avoid sacrificing the *nonproduct* quality viewpoints. In our project, which of *security, performance, reliability,* and so on are most important to us? What are we prepared to invest in this software? The quality we want to build into the product and the project is a negotiated compromise. In practice, this comes down to identifying the greatest number or level of *attributes* (product-based quality) that:

- We *can deliver* (we have constraints of money, time, resources, and skills);
- To *support the users'* tasks (user-based quality);
- While giving best *cost–benefit ratio* (value-based quality);
- While following *repeatable/quality assured processes within a managed project with minimized defects* (manufacturing-based quality).

The balance that we want to achieve between the quality viewpoints is defined, then formalized in the acceptance criteria that we set for the functional and nonfunctional attributes of the software product and project

deliverables (for example, specifications). Our acceptance criteria will prevent conflicts between stakeholders, provide guidance on how we develop the software, and mitigate against failing to meet the customer group's expectations.

Customers can find nonfunctional attributes difficult to conceptualize. As an example, in one project the customers had to set acceptance criteria for performance and response time. Initially they considered 20 seconds response time to be acceptable, even short. It *sounds* short. It *is* short when one is running for a train! However, the customers found it an unacceptable time *for a Web page to load* from the corporate intranet when they saw it live. The builders and supporters must be very proactive in helping the business define the nonfunctional acceptance criteria. Something that helped in the example above was a *performance prototype*; one of the team simply stopped the meeting for 20 seconds, allowing no speaking, no writing, and no movement, to demonstrate "what 20 seconds really feels like." You could also build prototypes and scenarios to help develop availability requirements and acceptance criteria.

Package vendors should note that customers often make purchases based on nonfunctional attributes, where functionality is similar in different products. Poor nonfunctional attributes may mean a product will be rejected [25]. Part of the market and other research for the project should explore this, with acceptance criteria set by the help-desk, support, sales, and marketing teams. Because the attributes are not described in the standard in everyday language, it can be useful to translate them into questionnaires, as suggested by Trienekens and van Veenendaal [26]. For example, asking "Have you an alternative way of carrying out your task if the software is not available?" provides an insight into *reliability* acceptance criteria. The weighting of replies for all the questions gives the priorities for acceptance [27].

The team should agree on the priority for the system attributes, because the most important attributes will be the focus for designing what is built. If the team then cannot set SMART acceptance criteria for the high-priority attributes, or cannot agree on the priority, try some prototyping to help generate ideas. Acceptance criteria should include the measurement of indicators. Indicators (for example, "percentage increase in market share") are measured after the project completes, so acceptance criteria should show which attributes contribute to the aims and, hence, to the indicators.

9.4.5.4 Outline plan

Once the aims and objectives, constraints and acceptance criteria have been agreed on, we can build an outline plan, based on the prototype plans we built before. The reason for doing this is to examine whether it is reasonable to provide a solution within the constraints that meets the aims, objectives, and acceptance criteria. This plan is unlikely to be very accurate or precise; expect to see "ball park" figures, and an allowance for replanning during any project.

9.4.5.5 Reviewing the contract

Any contract for further work should be reviewed by walkthrough, for understanding, and then by an inspection or similar review to identify defects. Whereas the walkthrough is mainly for sharing information and understanding, the inspection is a review that is focused on finding defects (see Table 9.5 and Appendix A) so both are needed to ensure we have the right contract. Each group will bring a different perspective to the review (Table 9.10). Note that a joint customer/supplier contract review is a requirement of ISO 9000, because the contract is the agreement and commitment to the work. It is important that the acceptance criteria for the software are reviewed, both for SMARTness and against the constraints, aims, objectives, targets, and indicators. Use a peer review or an inspection (Table 9.5 and [2, 17]), and ensure that each quality view is covered in the review. If the team cannot set, review, and agree on the contract including the acceptance criteria at least at a high level, then it is too soon to set the contract.

9.5 Exit from start-up stage

The exit from start-up happens either because we have decided to do nothing, or because we have a contract for further work. Examples of exit criteria are shown in Table 9.11. The specific exit criteria you use will depend on your organization. You may be feeding into some other planning process at this stage or you may go straight into a project, for example an SDLC. In some organizations, additional planning and authorization is required depending on the likely cost or risks of a proposed project.

Different organizations use different document names depending on the project management process. You may see the exit decision in a document called any of project idea form, project mandate, initiation document, project approach, or authorization to proceed. The important point is that it should document or refer to the output from all the steps covered in this chapter.

In this chapter, I have shown you some techniques that help you decide whether you need a SDLC and how to improve the contract for the work.

Table 9.10 Review Perspectives at Exit from Start-Up

Group	*Quality*	*Risks*	*Constraints*	*Aims, Objectives,Targets, Indicators, Acceptance Criteria, Outline Plan*
Customer	User	Impact on organization	Business, service level, time, cost	Realistic, time-bound
Managers	Value	Impact on other projects	Cost, time, skills, resources	Realistic, time-bound
Builder	Manufacturing, product	Likelihood—technical	Technical skills, knowledge, infrastructure	Achievable, realistic
Measurers	Product, manufacturing	Likelihood—previous failures, predictions	Technical skills, knowledge, infrastructure	Specific, measurable
Supporters	User, product	Impact on existing systems	Technical skills, knowledge, infrastructure	Achievable, realistic

Table 9.11 Example of Exit Criteria for Start-Up Stage

Example Exit Criteria	*Example Deliverables Documented, Reviewed, and Agreed Upon*	*Example Authorization Owners*
1. "Understanding the problem/idea" step complete	Problem statement, prototype assessments	Joint sign-off by representatives of all groups
2. "Decide whether problem/idea is worth solving" step complete	Problem assessment, cost–benefit analysis, risk assessment	Joint sign-off by representatives of all groups
3. "General constraints and parameters" step complete	Project proposal/brief	Joint sign-off by representatives of all groups
4. "Agree on next stage" step complete	Aims and objectives, constraints, acceptance criteria, outline plan	Joint sign-off by representatives of all groups
5. Criteria 1, 2, 3, 4 complete with *decision to do nothing*	Entry in suggestion scheme log	Process owner
6. Criteria 1, 2, 3, 4 complete with *high-level contract for SDLC*	Project mandate	Process owner
7. Criteria 1, 2, 3, 4 complete with *authorization for system change*	Initiation document	Process owner
8. Criteria 1, 2, 3, 4 complete with *authorization for nonsoftware solution*	Project idea form	Process owner

These techniques were summarized in Table 9.5. In the next chapter, I will describe the SDLC.

References

[1] Smith, K., "The Software Industry's Bug Problem," *Quality Digest,* 2003; reproduced on http://www.qualitydigest.com, April 2003.

[2] Gilb, T., and D. Graham, *Software Inspection,* Reading, MA: Addison-Wesley, 1993.

[3] Schein, E. H., *Process Consulting, Vol. 1: Its Role in Organizational Development*, Reading, MA: Addison-Wesley, 1988.

[4] TQMI, *Problem Solving—Tools and Techniques,* Frodsham, England: TQMI, 2001.

[5] Robson, M., *Problem Solving in Groups,* Aldershot, England: Gower, 1995.

[6] IEEE 1028™ Standard for Software Reviews, 1997.

[7] de Bono, E., *Six Thinking Hats®*, New York: Penguin, 1999.

[8] Evans, I., "The Troubled Project—Best Practice from Theory to Reality," *EuroSTAR Conference*, 2001.

[9] Hohmann, L., "Lo-Fi GUI Design," *Software Testing and Quality Engineering, 1,* 5, 24–29, September 1999.

[10] Nance, R. E., and J. D. Arthur, *Managing Software Quality,* New York: Springer-Verlag, 2002.

[11] Freeburn, G., "Mind Mapping 101 for Testers," *EuroSTAR Conference,* 2002.

[12] Buzan, T., *The Mind Map Book,* London, England: BBC Consumer Publishing, 2003.

[13] "Drawing Concerns: A Structured Rich Picturing Approach," http://business.unisa.edu.au/cobar/documents/richpic_colin.pdf, November 2003.

[14] Belbin Associates, "Belbin Team Roles," http://www.belbin.com/belbin-team-roles.htm, October 2003.

[15] Team Technology Web site, "The Mother of Strategic Systems Issues: Personality," http://www.teamtechnology.co.uk/tt/t-articl/news1.htm, October 2003.

[16] Honey, P., "Learning Styles," http://www.peterhoney.co.uk/product/learningstyles, October 2003. PeterHoney.com, 10 Linden Avenue, Maidenhead, Berks, SL6 6HB. Tel.: 01628633946. Fax: 01628633262. E-mail: info@peterhoney.com.

[17] McHale, J., "Innovators Rule OK—Or Do They?" *Training & Development,* October 1986; reproduced on http://www.kaicentre.com/, July 2003.

[18] Johnson, R., "Somebody's Watching You" *Sunday Times,* May 11, 2003.

[19] European Foundation for Quality Management, "EFQM Excellence Model," http://www.efqm.org, August 2003.

[20] IT Infrastructure Library, *Best Practice for Service Delivery,* London, England: IT Infrastructure Library, OGC/TSO, 2001.

[21] IT Infrastructure Library, *Best Practice for Service Support,* London, England: IT Infrastructure Library, OGC/TSO, 2002.

[22] Humphrey, W., *Introduction to the Personal Software Process,* Reading, MA: SEI, 1997.

[23] Humphrey, W., *Introduction to the Team Software Process,* Reading, MA: SEI, 2000.

[24] International Standards Organization/International Electrotechnical Commission (ISO/IEC), DTR 9126, Software Engineering—Software Product Quality (Parts 1–4, 2000/2001).

[25] Watkins, J., "How to Set Up and Operate a Usability Laboratory," *EuroSTAR Conference*, 2002.

[26] Trienkekens, J. J. M., and E. P. W. M. van Veenendaal, *Software Quality from a Business Perspective,* Dordrecht, the Netherlands: Kluwer Bedrijfsinformatie, 1997.

[27] Hendriks, R., E. van Veenendaal, and R. van Vonderen, "Measuring Software Quality," in E. van Veenendaal, *The Testing Practitioner,* Den Bosch, the Netherlands: Uitgeverig Tutein Nolthenius, 2002, pp. 81–92.

Selected bibliography

Belbin, R. M., *Management Teams–Why They Succeed or Fail,* London, England: Butterworth Heinemann, 1981.

Boehm, B. W., *Software Engineering Economics,* Englewood Cliffs, NJ: Prentice-Hall, 1981.

Obeng, E., "Helping Stakeholders to Understand Requirements," *Project Manager Today,* July 2003, pp. 14–17.

CHAPTER

10

Contents

Software-Development Life Cycle

In this chapter I shall:

- Describe the entry criteria, steps, and exit criteria for the SDLC;
- Outline and compare several approaches for the SDLC, especially for readers who are not software specialists;
- Identify the quality definitions from Chapter 1, which prevail at this stage;
- Identify how each of the groups identified in Chapter 2 become involved at this stage;
- Identify techniques that improve teamwork during the SDLC.

The users are moaning about the software; they say it is doing the wrong thing, but it is functioning according to the specification. When I saw the bug report, I marked it "Not a bug." They're just wasting my time!

—Irate builder, trying to get on with work

What do you mean it was not in the specification? You people wrote the specification. It's not my fault if you didn't keep it up to date!

—Irate customer trying to use the software

10.1 Software-development life cycle—description

The builders and customers are at loggerheads. Although the builders have developed what was specified, that is not what the customer really needs. Both are right from their own point of view, but they need a way to understand each other better in order that the customer's expectations meet the builder's

delivery. The project of building a software system to deliver to a customer is controlled via a software-development life cycle (SDLC). Some SDLCs are formally defined within methodologies. Others are informal but understood by those who use them—"We always do it this way." Regardless of the formality of the definition, all SDLCs are intended to deliver what the customer requires and to avoid the problems of our irate builder and customer.

In this chapter, I will compare models to see how well they do this. I will show you how the models used for the SDLC include detailed specification of what is needed, design of the solution, and the building and testing of products that will deliver what is needed.

In Figure 10.1, we see that the SDLC sits in the software life span between start-up (covered in Chapter 9) and delivery (covered in Chapter 11). During start-up, we identified a problem or idea, investigated it, and decided it was worth building a software solution, so we set a formal or informal contract of work for the SDLC. At the end of the SDLC, if we are successful, the software is delivered. It will be used postdelivery, as it is maintained and supported for the rest of its life span. The SDLC is not building to the moment of delivery, but to postdelivery when the software is used.

10.1.1 Types of software acquisition project

The purpose of an SDLC is to provide a software solution to help the customers solve the problems identified during start-up. The acquisition of the software to resolve a particular problem might be achieved in several ways, for example:

- *Custom-made system:* The whole software system is designed and built to meet a specific customer's requirements; the customer may choose to have the software built by a third-party supplier, that is, another organization, or to use an in-house development team consisting of by people who work for the customer's organization.
- *COTS system:* The customer buys a commercial off-the-shelf (COTS) system or package, sometime referred to as shrink-wrapped software, from a package vendor (a third party who sells COTS packages).
- *Tailored package:* The customer buys a package, but has some tailoring done to it by the supplier; these are changes made to meet specific requirements of the customers.

Any of these would be covered by an SDLC, but the customers will have more or less control over the detailed content of the software depending on the type of system chosen.

The customer may also decide to resolve their problem by requesting a change to an existing system. We will cover these maintenance changes in Chapter 12, but it is worth remarking that the same activities as in an SDLC

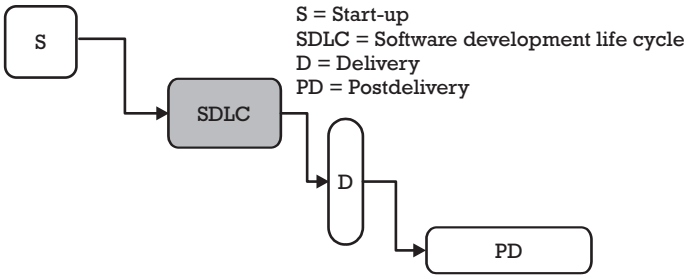


Figure 10.1 Life span stage diagram.

would take place in a maintenance change. The specification, design, build, and test of the change may be quite small activities, or may be equivalent to an SDLC, and, indeed, use the same life cycle model up to the point of delivery.

10.1.2 Identifying the software products

Remember that this is not just code being built by developers; people will also be busy designing, building, and testing training, documentation, and support materials. If the software being developed is a package for commercial sale, the support material may include sales and marketing material. For an in-house business system, it may include business process manuals. These are all part of the delivered product. When I talk about "software" or "product," I mean all of these types of deliverables. The media of delivery might be electronic, paper, on a microchip, or by semaphore signals; it does not matter.

10.1.3 SDLC task summary

There are several different life-cycle models for software development. In this chapter, we are interested in how the SDLC models help us achieve software quality, so rather than describing them fully, I am only going to outline them and give references to further information. The life-cycle models have changed over time to reflect the changes in customer group problems, perceptions of quality, and technologies available. Generally they have been described as development life cycles, but some of them include or may be adapted to maintenance activities. They have a number of steps in common. In some of the models we will see that these are performed once and in others some or all the steps are repeated or broken down into substeps. Put very simply, the fundamental steps are:

- *Planning and monitoring*—we plan what we will do and when we will do it, then track progress against our plan.
- *Managing change*—the real world will change around the SDLC and the deliverables required from the SDLC may need to change to reflect reality.

- *Requirements*—we make a detailed description of what we want.
- *Design*—we design the solution, including the software to meet the requirements.
- *Build*—we build the software based on the designs.
- *Testing*—we test to make sure that the software functions properly.

10.2 SDLC viewpoints

Just as we saw in Chapter 9 that start-up fails if the right people are not involved, during the SDLC we can have problems if we do not make sure that we take account of all viewpoints throughout the project. Sometimes, the customers set the contract, order the software, and then leave the managers, builders, and measurers to get on with it: "It's obvious what we want! Just do it!" This means that the customers are not involved in the SDLC. Similarly, the supporters may not be involved in the SDLC, only being "invited to the party" at delivery. If this happens, it limits the quality viewpoints (see Table 10.1, the definitions in Chapter 1, and assignment to groups in Chapter 2).

We saw in Chapter 2 that the builders and measurers tend to focus on manufacturing and product quality, rather than value or user quality. The manufacturing viewpoint focuses on delivering to specification and removing defects (differences between the product and its specification). The product viewpoint focuses on attributes such as functionality, performance, and security: how they are specified and built. Some builders and measurers, especially user-acceptance testers, do empathize with the user view, adopting a "fit-for-purpose" approach, but often these are people who are involved late in the SDLC, sometimes too late to influence design decisions.

Both builders and measurers hold a transcendent viewpoint ("In my heart I know what is right") that pursues technical excellence or achieving zero defects in software rather than cost-effectiveness or value. One colleague, an excellent tester, said to me when we discussed the value viewpoint, "But we must not sacrifice quality to cost." He was forgetting that although his manufacturing definition of quality focuses on removing defects, in the value view, keeping within costs is a major contributor to quality.

Table 10.1 Views of Quality During SDLC—Simple

	Group	
Quality View	*Builder*	*Measurer*
Transcendent view	✓	✓
User view		(✓ can be late in SDLC)
Value view		
Product view	✓	✓
Manufacturing view	✓	✓

✓ is a primary quality view.
(✓) is a quality view that may be taken by some people in this group.

When delivery takes place, the user and value viewpoints will prevail. The builders and measurers feel pleased that they have delivered to specification, and are then stunned to find the system rejected or criticized. Perhaps the specification was wrong, misinterpreted, or out of date. Perhaps an overambitious solution had been built, which means that the software has cost more or taken longer than budgeted. The customers, managers, and supporters turn on the others: "How could they have been so careless and extravagant?" The builders and measurers retaliate: "If you hadn't kept changing your minds…," and so on.

Much of this can be avoided by a careful start-up (see Chapter 9) but during the project some things will change (see Table 10.2 for examples) and unless all the groups are represented on the project, the SDLC may proceed without allowing for changes that will be necessary during the SDLC. We can deal with change in the SDLC provided we acknowledge that it is going to happen.

Getting everyone involved in the SDLC team will stop quality views being forgotten during the SDLC (Table 10.3) and will allow the changes in the organization's requirements to be reflected throughout the SDLC.

Table 10.2 Examples of Changes During the SDLC

Real-world changes	The customers' needs, hence the requirements for the software, will change to reflect their changing environment. These might be changes in the marketplace within which an organization operates, or legislative changes that affect how an organization may operate. If the SDLC team are not aware of these changes they will deliver unwanted software. As an example, the McCartney report [1] cites a project at an insurance company in the United States. By the time the project was finished, the company no longer sold the product to be supported by the software being delivered.
IT infrastructure changes	The hardware/network/system on which the software will be used may change, thus changing constraints and risks for the software. In one project I worked on, performance was a critical risk, but a hardware upgrade changed the performance constraints, thus changing the risk focus of the project from performance to other areas.
Existing systems (IT and business) changes	The new system does not "plug in" as anticipated, either because the builders were not aware of the detail of the IT environment and infrastructure or because it changed during the SDLC. The McCartney report [1] cites a government IT project in which changes to the existing systems during the SDLC made integration of the new system into the existing systems difficult.
Skills changes	People with expertise may leave or join the organization or the project, changing the project's capabilities and thus what could be delivered.
Leadership and organization changes	The organization's priorities may change, so the project may become more or less critical than it was, perhaps meaning a change in timescale, budget, people, or resources.
Risk changes	Risks change over time. One mistake project teams often make is to build a risk register at the start of the project and then never reassess the risks. Low risks at the start of a project may grow to come back and bite you later on: "Maybe the worst risk in a project is a lack of an ongoing risk assessment and risk management process" (David Hayman, personal correspondence, March 2003).
Errors made during start-up	We may realize we have made some mistakes during start-up—we are only human and we may have missed something. Building the solution we intended may be harder than we thought, we may have misunderstood an aim, or we may realize that acceptance criteria cannot be met within targets.

Table 10.3 Views of Quality During SDLC—Teamwork

	Groups				
Quality view	*Customer*	*Manager*	*Builder*	*Measurer*	*Supporter*
Transcendent view	✓	✓	✓	✓	✓
User view	✓			(✓)	✓
Value view		✓			
Product view			✓	✓	✓
Manufacturing view			✓	✓	

10.3 Entry criteria for SDLC

10.3.1 Entry criteria following a detailed start-up

If we have done everything suggested in Chapter 9, we arrive at the start of the SDLC with a contract and additional information gathered during start-up, which should include:

- Problem/solution analysis—"What problem are we trying to solve?"
- Constraints analysis—"What financial, time, organizational, and technical constraints do we have?"
- Cost–benefit analysis—"What benefits do we expect within our budget?"
- Risk analysis—"What is the likelihood that this will go wrong and the impact if it does or if we do not do it?"
- Aims and objectives for the SDLC—"Why are we doing this? What difference will this project make to the organization?"
- Targets and indicators—"How will we measure progress and whether the project has been worthwhile? Are we making a difference?"
- Acceptance criteria—"How will we know if we have succeeded?"

Table 10.4 Example Entry Criteria and Owners for SDLC

Example of Entry Criteria	*Example of Entry Documents*	*Example of Authorization Owners*
1. All start-up steps complete, deliverables documented, reviewed, agreed on, and signed off	Problem statement, prototype assessments, problem assessment, cost–benefit analysis, risk assessment, project proposal/brief, aims and objectives, targets and indicators, constraints, acceptance criteria, outline plan	SDLC project sponsor, process owner, project manager
2. Contract reviewed, agreed, and authorized	Documents from (1) completed, contract checklist completed, all documents received	Project manager
3. System change authorized	Documents from (1) completed and received, change mandate signed	IT support manager
4. Process change authorized	Documents from (1) completed and received, change mandate signed	Process owner

When we check the entry criteria for the SDLC, we should see all of this information in the contract and associated documents. Table 10.4 shows some simple examples of entry criteria for different projects. You may call some of these things slightly different names in your organization; that does not matter, as long as you have the information indicated.

10.3.2 When no entry criteria have been defined

If no entry criteria have been defined for an SDLC in your organization, you may find that you start work without all the information you need. In this situation, for your current project, whatever its size or status, I would recommend that you try to document something to describe the entry into the SDLC, even on the smallest SDLC. The virtue lies not in the size of the documentation, but in its clear content. At a minimum, take an approach of "on one page" to see if you can capture the essence of the problem in a checklist form like the one in Table 10.5 to give you a basic set of entry criteria. Use the checklist to see which areas you need to investigate more fully; for example, do you have clear acceptance criteria and constraints?

To prevent this problem from recurring, improve the SDLC model that is used in your organization by adding entry criteria to the model.

10.3.3 When entry criteria have not been met

Strictly, if the entry criteria have not been met, we must not start the SDLC. In order that we can be strict, it is important that the entry criteria are:

Table 10.5 Checklist of Entry Criteria Form for a Small SDLC

Project name, date of form completion					
Problem/solution analysis—Do we understand what problem are we trying to solve? Is that understanding shared by all parties? Problem statement:					
Constraints analysis—Have the financial, time, organizational, and technical constraints been defined and agreed on? Constraints statement:					
Cost–benefit analysis—Do we understand our expected benefits within our budget? Cost–benefit statement:					
Risk analysis—Have we analyzed the likelihood that this will go wrong and the impact if it does or if we do not do it? Likelihood: Impact of doing it wrong: Impact of not doing it:					
Aims and objectives for the SDLC—Do we understand why are we doing this? What difference will this project make to the organization? How will we measure progress against the aims and objectives? Aim and objectives of the SDLC: Targets and indicators of success:					
Acceptance criteria—How will we know if we have succeeded?					
Agreement Signed Date	Customer	Builder	Manager	Measurer	Supporter

- Objective;
- Defined in a way that is SMART (specific, measurable, achievable, realistic, and time-bound);
- Necessary rather than simply desirable.

However, there are occasions when we might decide to waive the entry criteria. This may happen if we decide that the entry criteria are excessive or incorrect, or if we believe we are faced with an emergency. There are also occasions when we may wish to increase the entry criteria, for example, if we see that the SDLC is at a higher risk than our normal projects.

10.3.3.1 We decide the entry criteria are excessive or incorrect

If the entry criteria include items that are desirable rather than necessary, we may wish to redefine them. This requires careful thought; if you do not have all the entry documents or you do not have an authorization to start, what risks are you taking? If the decision is made to bypass or reduce the entry criteria, do we understand the risks we are taking? Is the risk acceptable? I would recommend that you document the reason for deciding that the entry criteria can be reduced or ignored, as part of your project documentation. This might be in the quality plan, which typically documents the quality gates for a project. If the defined entry criteria are incorrect, we should redocument them, again with an explanation, and, if possible, suggest a process improvement that would prevent us from making the same mistake again. Use the checklist in Table 10.5 to help you do this.

10.3.3.2 It's an emergency

As we saw in Chapter 1, in CMM® [2] one of the differences between high-maturity and low-maturity organizations is how the organization reacts to change or crisis. A high-maturity organization holds onto processes during change and crisis, as this helps to manage the problems. A low-maturity organization will discard the processes during emergencies, believing that they add bureaucracy rather than value. This is counterproductive, especially with the entry into an SDLC. The processes should be defined at a suitable level of control without unnecessary bureaucracy; if a metric or a piece of paperwork does not facilitate control, do not have it in the process. Without clear constraints and acceptance criteria, it is difficult to know when to stop work in the SDLC; we have nothing on which to base our exit criteria. In an emergency, this means that we may forget to do something vital. I would recommend that an emergency request still goes through entry criteria, because in an emergency we need processes to support our decisions; we are more likely to make mistakes if we are doing something in a hurry. Some organizations define an emergency SDLC, which encapsulates the spirit of the controls on a single checklist. This can be a useful approach.

10.3.3.3 We decide the entry criteria are insufficient

If we perceive high risk in the SDLC, we may decide that the defined entry criteria are not stringent enough. In this case, we should add additional criteria, or increase the "pass level" for the entry criteria. We can do this if we allow tailored entry criteria.

10.3.4 Tailoring entry criteria

In several organizations I have observed, project managers have available a number of agreed-on SDLC templates, and they select an SDLC with suitable entry criteria, to meet the specific risk levels for their own projects. Suppose that a particular organization decides on the stringency of the entry criteria for an SDLC based on three factors, the estimated cost of the work, the estimated size in days, and the perceived risk. Each factor is assessed on a score of 1 (low) to 3 (high). The factor scores are multiplied together to give an overall score that will be between 1 (1 · 1 · 1) and 27 (3 · 3 · 3). The score is used to pick entry criteria levels from a list like the one in Table 10.6.

Table 10.6 Example Tailored Entry Criteria

Cost · Size · Risk Score	*Example Entry Criteria* *Example entry documents*	*Example authorization owners*
1–5	1. E-mail with request for change from customer, copied to manager, builder, measurer, and supporter 2. Entry checklist in Table 10.5 completed Deliverables: E-mail and acknowledgment, entry checklist in Table 10.5	Customer, builder
6–12	1. As above, plus 2. Request raised at monthly change planning meeting 3. Additional problem statement/assessment, constraints, acceptance criteria documentation completed Deliverables: As above, plus additional notes on problem statement/assessment, constraints, acceptance criteria	Customer, process owner, builder, manager
12–20	1. As above, plus 2. Acceptance criteria prioritized and documented with metrics from ISO 9126 3. Solution prototyping Deliverables: As above, plus additional work on cost–benefit analysis, constraints, acceptance criteria documentation	Customer, process owner, project manager
20–27	1. As above, plus 2. Additional risk management planning carried out by process owner and project manager 3. Project authorization board has discussed and agreed to the change and risk plan Deliverables: As above, plus additional work on problem statement, prototype assessments, problem assessment, cost–benefit analysis, risk assessment, project proposal/brief, aims and objectives, targets and indicators, constraints, acceptance criteria, outline plan, project authorization	Project authorization board, project sponsor, process owner, project manager

10.3.5 When no start-up stage took place

It may well be that you are at the start of an SDLC, or even partway through it, and you do not have a detailed start-up as described in Chapter 9. What do you do?

- *Start of SDLC—Large:* If you are at the start of a large or high-risk SDLC, I would recommend that you carry out the activities in Chapter 9 before proceeding, in order that you can plan the SDLC.
- *Start of SDLC—Small:* If you are at the start of or working on a very small SDLC, you may feel you do not need formal documented entry deliverables, but you will need the information content of the deliverables, documented or in the team's heads. You will need this to plan the project, and to control it, especially when the going gets tough. Use the checklist in Table 10.5 to help you decide whether you have enough information, or whether you should investigate fully.
- *Midway through SDLC:* If you are midway through an SDLC as you read this, you may be wondering how to apply these ideas: "Shouldn't the customer and manager have sorted this out when the contract was signed?" Yes, perhaps so, but if they have not, or if the information you have is not full enough, you may have to revisit the start-up activities. You will need acceptance criteria in order to understand if the software is ready for delivery, for example, and you will certainly need to know your constraints. Use the checklist in Table 10.5 to help you decide what you need to find out, in order to complete the project.

10.4 SDLC—typical activities

10.4.1 Planning and monitoring

It is essential to have a plan for the SDLC. The plan must be based on the information identified during start-up. For most organizations, there will be constraints on cost, time, resources, and skills; we will not have an infinite budget. The acceptance criteria identified at start-up need to be met within the constraints.

The entry criteria give us enough information to start planning. The plan needs to describe how the SDLC will deliver within the constraints, meeting the acceptance criteria and containing the risks, to meet the aims of the project. We will also need to break down the plan into smaller, more detailed plans (bite-sized chunks). One project manager I know assigns responsibility for detailed planning of each chunk to people within the team saying, "Tell me how you will deliver what you need to do within this timescale." This is very effective as it allows each team within the project to understand their constraints and to own their own plan.

Once the plan is put together, it needs to be reviewed and agreed on. The Team Software Process (TSP) [3] recommends a "launch" at the start of the

SDLC, where the team goes through the plan and commits to it. A shared planning process followed by a launch meeting allows people to discuss the plan, share causes for optimism and pessimism, understand each other, and so commit to the plan and the SDLC team wholeheartedly.

The plan is used throughout the SDLC to track, measure, and control progress. We inevitably replan during the SDLC as we learn more; it is worth designing replan points into the SDLC and acknowledging those in the contract of work.

Planning and control need to take place at different levels of detail for different people; we need to understand what we do today to be on time at the end of this week for our team, in order to meet that major project milestone in three months. This will match with reporting requirements between the groups, as we saw in Chapters 3 to 7. Managers carry out planning and control with the help of builders, measurers, customers, and supporters. We saw in Chapter 4 that this is needed on a day-to-day basis so that we manage our time to complete tasks today that contribute to meeting an end date months away. We also saw how dependencies between tasks mean they may be on the critical path—if they are not completed on time, then the whole project becomes late. The plan will need to be reviewed; all the groups need to agree that it is a plan to which they can deliver. In particular, there will be intergroup dependencies on the plan—everyone needs to understand and commit to these.

10.4.2 Managing change

During the SDLC, things will change and this will affect the plan and all the activities of the SDLC. One project manager I know regards his plan as "A basis for making changes"; he says "The plan is a description of what will not happen in the future." We saw in Section 10.2 that we will have to deal with changes in circumstance during the SDLC. These types of changes will affect the SDLC in different ways.

We will see later in this chapter that some of the SDLC models are more amenable to managing change than others, but here let us consider which SDLC activities are particularly affected by different types of change. Table 10.7 shows the most affected areas for each type of change. The changes may affect any of the activities; for example, the plan may be

Table 10.7 Example of SDLC Activities Affected by Example Change Type

	Most Affected Areas				
SDLC Activity	*Plan*	*Requirements*	*Design*	*Build*	*Test*
Real-world changes			✓	✓	✓
IT infrastructure changes			✓	✓	✓
Existing systems (IT and business) changes			✓	✓	✓
Skills changes		✓	✓	✓	✓
Leadership and organization changes	✓	✓			
Risk changes	✓				✓
Errors made during start-up	✓	✓	✓	✓	✓

affected by any of the changes, and leadership changes may affect any of the areas, but in the table I have indicated the most immediate effects. Some of these cause a chain reaction; we can see that real-world changes may affect the customers' requirements; if these change, then the design, build, and test of the software will also change. Changes to risk will affect the plan and specifically will affect the focus for testing.

If the SDLC model we choose is very rigid, then, superficially, it can appear easier to plan and manage it. However, the rigidity in the SDLC and the contractual arrangements around it make it very difficult to allow changes, whether these have arisen from errors we have made or from real changes in circumstances around the SDLC. This means we are in danger of delivering the wrong solution. We might deliver the wrong product attributes, or deliver solutions with manufacturing defects, at reduced value to the organization, with reduced usefulness, damaging attitudes to the team or the chosen solution.

A more flexible SDLC model, designed to allow for change, removes the problem of rigidity in the solution, but is more difficult to manage because everyone (customer, manager, builder, measurer, and supporter) needs to allow for uncertainty and change in the plans. For example, in the TSP [3], Watts Humphrey suggests a multiphase approach with a "relaunch" meeting for the team at the start of each phase.

10.4.3 Requirements

Some statement of the customers' requirements is needed. The supporters and the managers will also have requirements. The basis of these is the problem/solution analysis and the acceptance criteria defined at start-up. During requirements definition these are explored in much greater depth. The builders will document the requirements; often, this will be done by specialists in analysis. The customers, managers, and supporters all contribute to the acceptance criteria and hence to the requirements (see Chapter 9). The requirements will need to be reviewed. The customers, managers, and supporters will need to review that the builders have a correct understanding of the requirements. The builders, who have not directly been involved in requirements gathering need to know that the requirements have been defined in a way that allow them to design and build the software. The measurers, including testers, want to know that the requirements are testable; that it will be possible to measure whether the requirements have been met. The requirements will include:

- Software/system requirements;
- Infrastructure, operational, and support requirements;
- Training requirements (for software, support and business processes);
- Documentation requirements (for software, support, and business processes);
- Business process requirements (manual processes, for example);

- Implementation requirements (for example, media, time constraints, phased or big bang, ability to roll back) [4];
- Data transfer requirements; for a data migration this might include the contents of databases and standing.

All of these will be needed for the solution to be implemented.

10.4.4 Design

The solution will need to be designed. The design is based on the requirements. Some life cycle models divide this into several design areas. You will find that different models and methods have different names for the design stages, but there are three main design steps. An overall business-process design looks at the processes within which the customers will use the system, and would include design of training, documentation, and business processes. System architecture is the design of the overall software system. A detailed technical design covers the design of individual programs within the software. In this chapter, for ease of explanation, I have grouped these together as *design.* Builders will carry out and document the design. These designs will need to be reviewed. The supporters' acceptance criteria directly affect the design, for example, for performance, memory management, and maintainability requirements (see Chapter 9), and so they will want to review those designs. Customers may become involved in reviewing screen and report designs, both functionally and for usability. Builders need to know they will be able to build the software from the designs. Measurers want to know that the designs are testable; that it will be possible to measure whether the design goals have been met. Managers will want to review the designs for cost-effectiveness and value, for example, reduction in future maintenance costs.

The design will include:

- Software/system;
- Infrastructure, operational, and support processes and equipment;
- Training (for software, support, and business processes);
- Documentation (for software, support, and business processes);
- Business processes (manual processes, for example);
- Implementation processes, including rollout and rollback plans;
- Data conversion design.

10.4.5 Build

The solutions need to be built. What is built and how it is built is based on the requirements and design. This includes writing code, but also building all the other software products. Training material, user guides, business process manuals, marketing material, and support manuals all need to be built. All these software products can be reviewed against acceptance criteria.

Supporters review the code and support material. Customers review training, user, process, and marketing material. Measurers will review all the software products for testability and to prepare for testing. Managers will review progress and costs. Builders will take part in those reviews, to cross-check the different types of product and to check adherence to standards.

The build will include:

- Software/system;
- Infrastructure, operational, and support processes and equipment;
- Training (for software, support, and business processes);
- Documentation (for software, support, and business processes);
- Business processes (manual processes, for example);
- Implementation processes, including rollout and rollback plans;
- Data conversion software.

10.4.6 Testing

The software products need to be checked. As with design, there are several types of testing, and they are named differently in different models and in different organizations. Typically, models allow for four levels of dynamic testing. This is testing where code is executed. The lowest level of dynamic testing is often called unit, program, component, or module testing; the building blocks of the system are tested individually as they are built. Next, the building blocks are linked or integrated together, and the links are tested; typically, this is known as integration, link. or string testing. These two levels of testing will be based on the design and the code. Once all the building blocks have been integrated together, system testing takes place; the system as a whole and the processes using it are exercised. This level of testing is based on the requirements and the design. It may also be necessary to test how this system interacts with other systems. Finally, acceptance testing takes place; this checks the system against the acceptance criteria to decide whether it can be delivered. Various types of tests are run at each level, to check functional and nonfunctional attributes (see Chapter 9). Specialist testers will be jumping up and down at this point and shouting, "Testing is not just done at the end of the SDLC!" That is absolutely true; we have opportunities to test statically as well. This is testing done without executing the code. It includes all the review activities we did during start-up, planning, review, design and build, plus static analysis, by which we examine the code for flaws without executing it. These are all measurement activities. As we saw in Chapter 6, people from any of the groups may join the measurers group on a temporary basis. The test designs and the test results should be reviewed. The choice of people to carry out the review depends on the level of testing; any of the groups might be involved.

Depending on the scope of the project, the test stages might include testing of:

- Software/system (reviews of products, component, integration, system, and acceptance);
- Infrastructure, operational and support processes and equipment (reviews of products and operational acceptance);
- Training (reviews of products, trial runs, and walkthroughs);
- Documentation (reviews of products, trial runs, and walkthroughs);
- Business processes (review of products, walkthroughs, acceptance testing);
- Implementation processes, including rollout and rollback plans (review of products, walkthroughs, acceptance testing);
- Data conversion process and software (conversion software testing, data checks, dry runs).

10.5 Entry and exit points within the SDLC

There are entry and exit points between the steps; sometimes, these are called quality or project gates. Depending on the SDLC model that the project uses, these may appear at different points. As we will see, some life-cycle models have a strict cutoff between steps. In others, steps are omitted, repeated, or overlapped. The strictness of the criteria depends on the level and type of risk for the project as well as the skill set of the people on the team. In some SDLC models, particular handover points are regarded with significance and very formal entry and exit points are used. These are sometimes referred to as quality gates or project gates [1]. They are often used where control of the work is handed from one team to another, to ensure that all the deliverables have been handed over and are fit for purpose.

An example of a simple SDLC is found in Figure 10.2, showing the minor entry and exit points, indicated by an arrow between activity rectangles, and the major quality gates, indicated by the rounded rectangles. For a simple situation, if no training material, business processes, or infrastructure changes are required, this may be enough. Figure 10.3 shows a more detailed SDLC. Here the major quality gates are shown, and the activities between them explicitly include all the products and activities types, as well as differentiation between possible exit routes—to deliver a product or cancel the project, for example.

10.6 SDLC models

People talk about big bang, phased, iterative, and incremental approaches, but not everyone uses the words in the same way. In what follows, I will use building a model village as an analogy for how the terms are used in the descriptions below. The steps taken in building this village are set in italics. First, I will give an outline of the models, with the advantages and

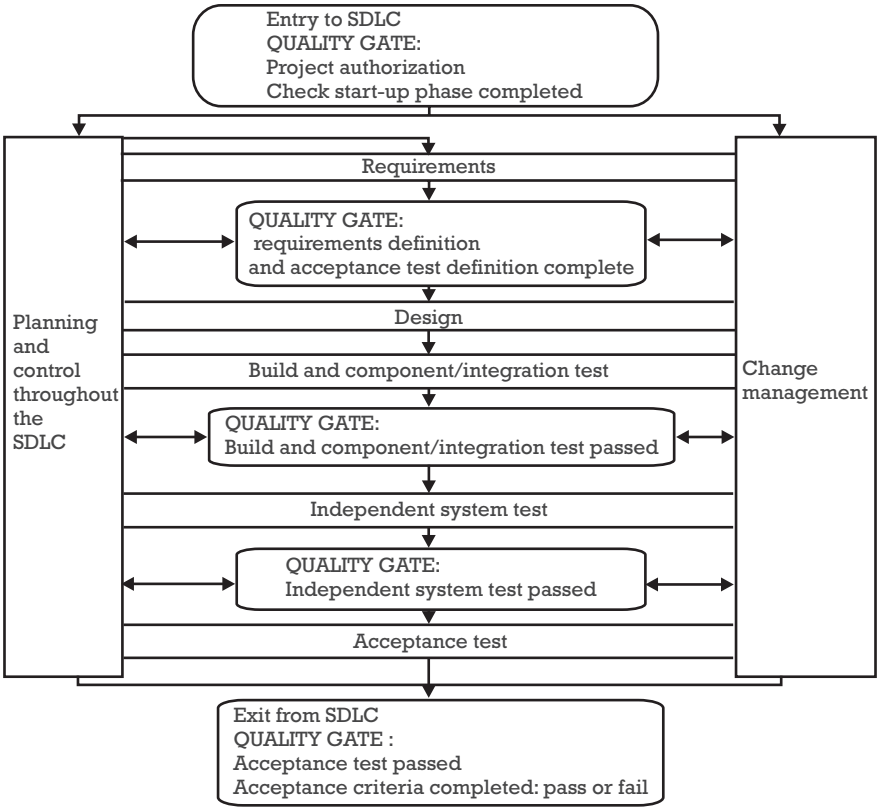


Figure 10.2 SDLC with simple entry, exit, and quality gates.

disadvantages of each model listed in Table 10.8. Then I will show how to tailor the SDLC models. The published SDLC models I have looked at in writing these outlines are listed in Table 10.9.

10.6.1 Waterfall model (big bang or phased)

This is the most traditional of the models. It is used on either a big bang or a phased project. The requirements are set at the start of the project, and all subsequent designs, code, and other products are based on those defined requirements (see Figure 10.4). Each step has a clear cut off; the requirements are agreed on and frozen, then the design, and so on; water only goes one way down a waterfall. An example of a defined waterfall model SDLC from DoD-STD-2167 is in [5].

10.6.1.1 Big bang waterfall

A project may be run with a "big bang" approach. *All the work is delivered at one time; all the houses, roads, and gardens must be ready at the same time for the village to open.*

At the start of the SDLC we can make a clear statement of what we need. Customer: "These are our requirements." Manager: "I will need three

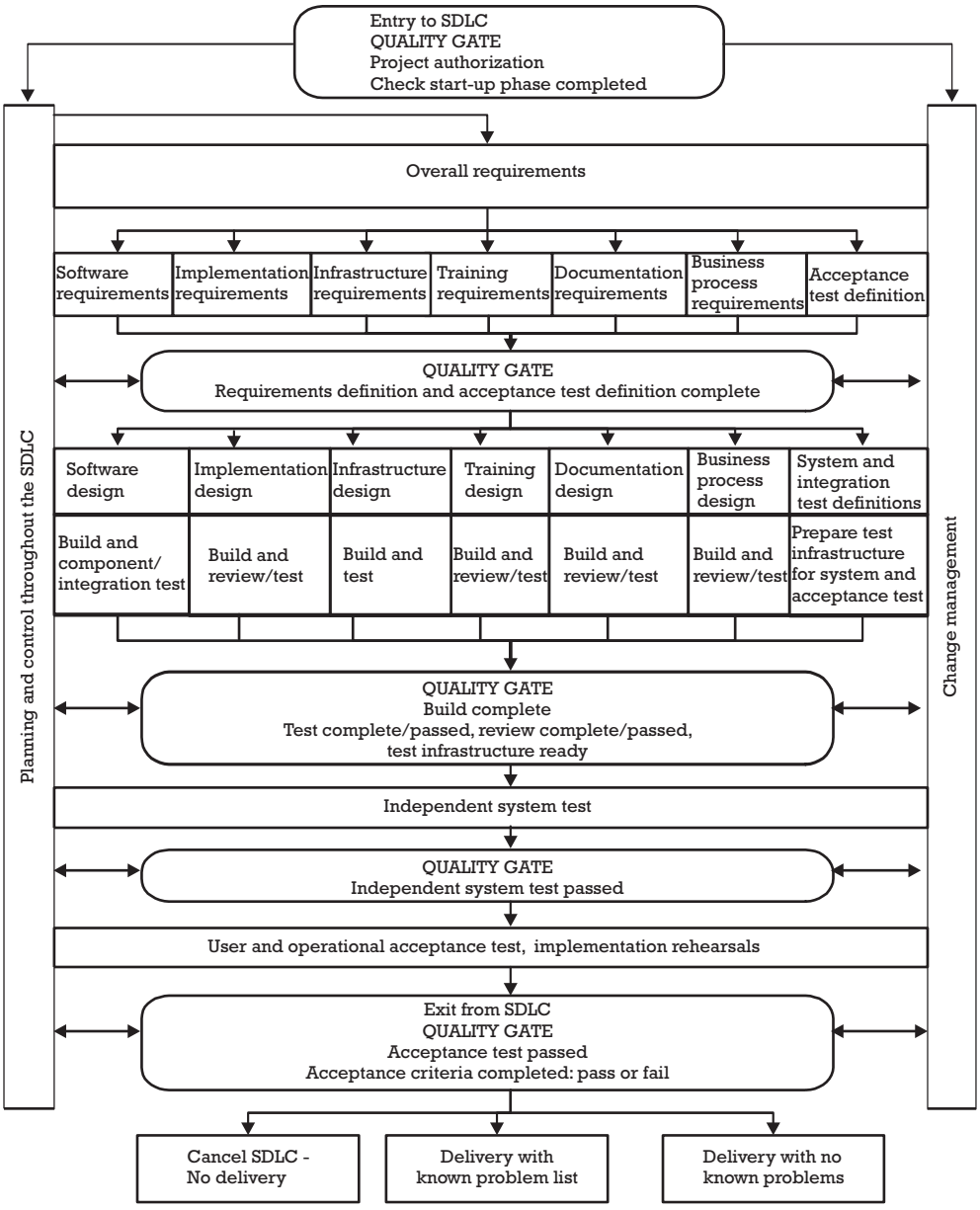


Figure 10.3 SDLC with detailed entry, exit, and quality gates.

analysts for 6 weeks, then five designers for 10 weeks, then 15 programmers for 10 weeks, then five testers for 6 weeks." After the requirements sign-off, measurers, customers, and supporters do not become involved again until testing.

10.6.1.2 Phased waterfall

In a project that has a "phased" approach, the work is grouped into separate chunks and delivered at intervals perhaps by different teams. *The village is*

Table 10.8 Advantages and Disadvantages of the Models

Life Cycle	*Advantages*	*Disadvantages*
Waterfall—phased or big bang	Clear-cut steps make project management and resource management easier. Audit trail.	One chance to get it right, mistakes in requirements and design are not found until testing; costly to repair. Does not allow for changing requirements.
	ISO 9000 compliant	
Incremental	Enables larger problems to be tackled in small chunks by one team.	Does not, unless combined with iteration or spiral or evolutionary, solve the waterfall problems.
Iterative/spiral	Allows for change and high customer involvement	Time for test and regression test increases. Long time between start and going live. Some practitioners say not so clear-cut for management by stages, could be harder to control time and costs, less clarity on milestones.
	Can be time-boxed (value view—cut losses if not productive).	
	Some practitioners say it is easy to manage time. Allows refinement of plans and ideas.	
	Testing earlier—faults found earlier. Do important areas first.	
Evolutionary	As iterative/spiral, plus bite-sized milestones, short time between start and going live, deliver important areas first.	As iterative/spiral, plus risk of faults going live sooner if insufficient testing.
V-model	Easy to manage; clear contract sign-off points, meets ISO 9000:1994. Test is continuous and cost-effective as defects found and fixed earlier.	Resourcing different, early weight to life cycle can seem bureaucratic. Can feel costly early in the life cycle.

Table 10.9 SDLC Models Referred to in This Chapter

Example of the SDLC Model	*Comment*
Boehm's Spiral Model [5, 6]	Uses prototyping and replanning with reassessment of risks and constraints with each prototype.
Team Software Process (TSP) [3]	Emphasis on clearly defined team roles and goals, checklists of activities, measurement, use of CMM® Level 5 processes. Phased or iterative approach.
Giddings Domain-Dependent Life Cycle [5]	Waterfall model with feedback loops, and experimentation or prototyping.
Rational Unified Process (RUP) [7]	Based on the spiral model, uses iterations within increments, with testing happening throughout the SDLC. Emphasis on defining, building, and testing the most important parts of the software first.
eXtreme Programming (XP) [8]	Incremental approach, relies on continuous (automated) testing by builders, on oral rather than written communication, and close collaboration.
Dynamic Systems Development Method (DSDM) [9]	Phased or incremental approach, with strong emphasis on people, their skills, and their ability to work in teams.
Gilb's Evolutionary Model [10]	Incremental or phased approach with delivery at the end of each increment.
DOD-STD-2167 Model [5]	Classic waterfall model.
V- and W-model [11, 12], Component Test Process Life Cycle [13], STEP (Systematic Test and Evaluation Process) [11]	Life cycles that emphasize the place of and control of software testing; these may be considered as fitting with the other life cycles. In this chapter, I have concentrated on the V-model and how it might fit in the SDLC models.

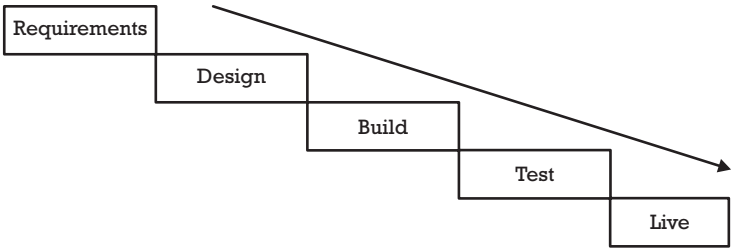


Figure 10.4 Waterfall model—big bang.

divided into four sectors (N, S, E, and W) that are worked on as separate projects. The whole of a sector is delivered together as one project, but each sector might be ready on a different date. It is possible to divide a waterfall style into phases, which run in parallel, provided that the work can be divided into distinct chunks, each being smaller, more manageable miniprojects with its own team.

10.6.1.3 When to use the waterfall method

The waterfall is a good model to use if:

- The contract for the work is complete, correct, and unambiguous.
- The requirements and acceptance criteria are complete, correct, and unambiguous.
- The work can be completed within the constraints.
- No change is expected in the requirements or design—the customers' world is static.
- The problem and solution are well understood and clearly defined.
- No one is going to make mistakes in the requirements or design.

It is very rare (I am tempted to give a probability of approaching zero) for all these to be true for a software development; software is too complicated, the world is constantly changing, and people are fallible. We need something better.

10.6.2 Spiral, incremental, and iterative models

There is a large family of incremental and iterative models that were developed from the 1980s onwards as a response to the problems of the waterfall model. These include increments in miniwaterfalls and in iterative or spiral models, as well as evolutionary models.

10.6.2.1 Incremental

In an incremental project (Figure 10.5), the work is chunked in a different way from a phased model. *The work is divided into houses, roads, and gardens. The roads are put in place, then the houses, and, finally, the gardens. The*

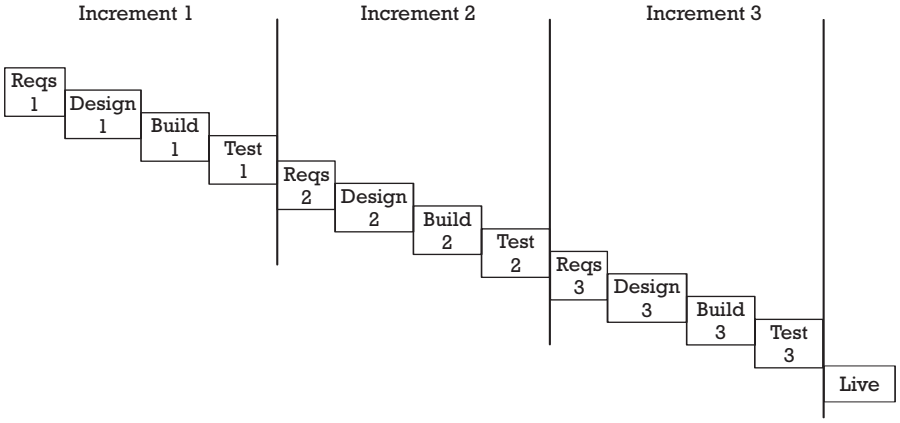


Figure 10.5 Incremental model.

incremental model is similar to the phased model, but one team can work on all the increments, whereas in the phased model, separate teams deal with each phase in parallel. This approach is useful if:

- The problem is too large for the team to tackle at one time, so building in stages is sensible
- The solution must to "go live" all at once

10.6.2.2 Iterative and spiral models

In an iterative approach (see Figure 10.6), steps in the project are repeated (iterated) allowing work to be revisited and refined. *The most important roads, with houses and gardens, are put in place first. As new houses are built, additional roads and gardens are added, and existing structures may be altered to fit with the new areas.*

The spiral model [5, 6], as shown in Figure 10.7, uses a series of prototypes, and through the prototyping refines our understanding of what we

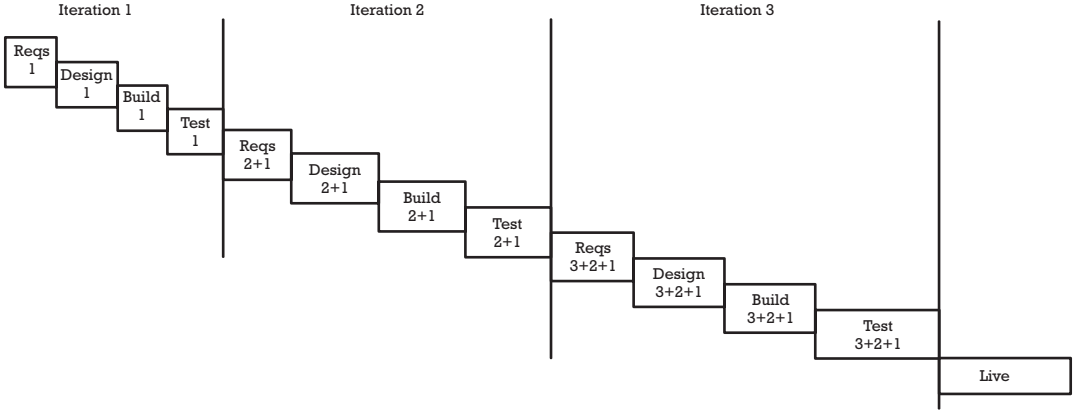


Figure 10.6 General iterative model.

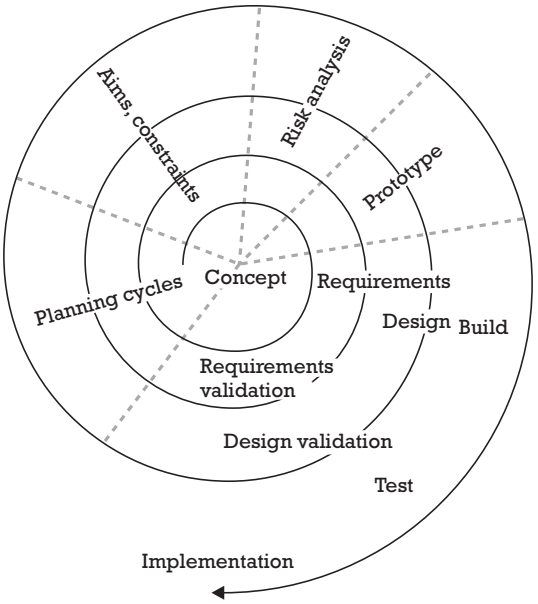


Figure 10.7 Spiral model. (*After:* [5].)

want to build, and our understanding of the risks, constraints, and so on. With each circuit of the spiral, we refine our plans against our improved understanding. *We build a paper plan of the village, then a cardboard scale model, then a full-scale model. As a result, we refine our understanding of what is required in each house, garden, and road, and we are able to cater for changes in requirements*.

The iterative and spiral models are more exploratory than the other models; they suggest that the same areas of requirements, design, build, and test are revisited repeatedly during the project, in order to correct errors, refine understanding, and introduce changes. However, each iteration will take more effort and time, to ensure that changes to existing structures are defined, made, and tested properly, and that unchanged areas have not been adversely affected. This is known as retest and regression test. The ability to change requirements may be useful for the customer, but is difficult for managers and builders if a fixed-price contract is agreed. As a colleague remarked to me, "Changing requirements can be an advantage and a disadvantage. If you have bid a project very thinly in a competitive tender process, the last thing you want to do us allow the customer to change requirements. It can make the difference between making money on the project or losing it." In particular, the spiral model encourages the use of prototyping (see Table 10.10 and Appendix A) as a technique to clarify what is required and to try out different ideas. It is useful because it acknowledges that we may not know everything we need to know, particularly about timescales and budget, at the start of the SDLC. Additionally, validation (does the specification describe what the customer needs?) and verification (have we met the specification?) take place at each step. In the TSP [3], for example, a "relaunch" of the plan is carried out at the start of each iteration.

Table 10.10 Summary of Techniques for Teamwork During the SDLC

Subject Area	*Technique Examples*	*In 30 Seconds… (see Appendix A for more)*
Team relationships and natural roles/team skills	Belbin team scores [14]	The SDLC can fail because of personal rather than technical factors. Teams need to understand their strengths and weaknesses as a team. A balance of roles/skills is required in the personalities in the team. Example roles: plants have new ideas; completer–finishers want to finish to fine detail. Too many plants and you will never finish anything.
Improve communication—empathy with others	MBTI [15]	Different people have different personalities and communication styles. People who wave their arms around and talk a lot can annoy people who like to be quiet and think, and vice versa. The Myers-Briggs Type Indicator (MBTI) identifies four contrasting type pairs (e.g., Introvert/Extrovert), leading to 16 "types" (e.g., INTJ is Introvert-iNtuitive-Thinking-Judging).
	Honey & Mumford [16]	The Honey & Mumford Learning Styles Questionnaire identifies preferred learning styles (e.g., Pragmatists and Theorists require different experiences to learn).
	Kirton [17]	Kirton identifies preferred problem solving methods (Adaptors versus Innovators)—do we break the rules or work within them?
Improve meetings	De Bono's Six Thinking Hats [18]	Improve meetings by setting rules for behavior. Six "hats" are used. Everyone wears the same color hat at the same time: Blue Hat—meeting structure, Black Hat—pessimistic, Yellow Hat—optimistic, Red Hat—feelings, White Hat—facts, Green Hat—creative ideas. Allows meeting members to move outside their stereotypes and allows time for different, sometimes difficult types of communication.
Identify problems and root causes, find solutions	Ishikawa fishbones [19, 20]	Use to identify problems, root causes of problems and solutions. On a fishbone diagram, brainstorm problems, their possible causes, their root causes, and, therefore, solutions to the root cause.
Review documents and other products	Reviews [21, 22]	There are five types of review: management review, technical review, inspection, walk-through, and audit—all of which are relevant throughout the SDLC. Specialist testers regard them as a form of testing because they are used to find and prevent defects in products and processes.
Understand/explore ideas	Prototyping [5, 23]	Not an SDLC! We prototype to try out ideas or, if we are not sure what we want, we build a model of a possible solution(s). There are two types of prototyping in software, "lo-fi" and "hi-fi." In hi-fi (high-fidelity), we build screens like those the customer will see. Lo-fi (low-fidelity) prototyping uses paper/white board. Discuss the prototype and review it. There is no danger of believing it is the software.
	Modeling and picturing, stories, metaphors [24–27]	There are a number of modeling and picturing techniques that can be worth exploring with lo-fi prototyping. These include Rich Picturing and Mind Mapping. Sketching a picture of a problem, solution, or idea can clarify it. Use stories, metaphors, and analogy as well; they can aid in understanding.
Understand whether an idea is worth pursuing	Risk workshop by brainstorming [19, 20]	A brainstorming workshop is run to list all the possible risks people can identify. The risks are sorted into groups, separating risks (might happen in the future) from issues (problems right now) and constraints. The risks are scored for impact and likelihood, and ranked to give a prioritized list.
	Cost–benefit analysis [19]	Cost–benefit analysis is done by calculating the predicted benefits of the proposed change (time, money saved) and setting this against the predicted cost.
Track progress	Earned value [28]	Not only is it possible to track cost against budget, but also what the cost so far was supposed to achieve compared with what actually has been done.

One problem that has occurred with the use of the iterative models is the commonly held belief among managers and customers that testing is reduced, for example, by removing test levels or by reducing the time to test. This is not the case. A colleague notes: "The clear implication of iterative methods is that you start testing earlier and test more often through the life cycle, not to mention the regression testing of all the earlier work; ergo, you do more testing." For example, in eXtreme Programming (XP) [7], the SDLC has two levels of testing (programmer and user) instead of four. I have received mixed reports of how well this works; XP enthusiasts say it works well and streamlines the project, but some testers are reporting differently:

> The theory is very plausible-plan and design test before development. Simplified roles and responsibilities, simplified testing phases. The reality … is very different. In practice, testing seems to fare even worse under an XP project than a traditional one (in terms of being neglected or done poorly).

10.6.3 Evolutionary model

We can see that the incremental, iterative, and spiral models all still suffer from the problem of the length of time from the start of the SDLC to delivery of software for use by the customers. The evolutionary model (Figure 10.8) was developed, for example, by Gilb [8], because it helps to break down the software into chunks that can be delivered earlier to the customer; this means that the real-life problem is at least partly resolved more quickly. According to Tom Gilb (at a seminar on Evolutionary Delivery, London, England, September 20, 2003), two characteristics that mark an evolutionary delivery as opposed to either an incremental or a phased delivery are the very large number of small increments and deliveries and the emphasis on feedback loops at each delivery so that continuous improvement is built into the SDLC. *The first deliveries of the village put dirt roads in place, plus some houses with basic rooms complete and the gardens laid to grass, in order that the first families can move in. In the later deliveries, the roads are asphalted, additional road furniture is installed (benches, night and security lighting), play and garden rooms are added, spare bedrooms fitted out in the existing houses, and new houses are added. As deliveries progress, each provides some new improvement: gardens are relaid according to individual requirements for flower beds and paths, garden sheds are installed, water features and other refinements are added to the gardens, and the kitchens are refurbished.*

10.6.4 V-model

The V-model and its close relation the W-model [12] were developed by testers who wanted to emphasize the cost-effectiveness of early testing. V-model SDLCs explicitly describe review activities as early testing. They expect that specialist testers are involved from requirements definition onward. *In the village, buildings inspectors are involved from the planning stage onwards, as are representatives of the people who will "accept" the village as suitable*

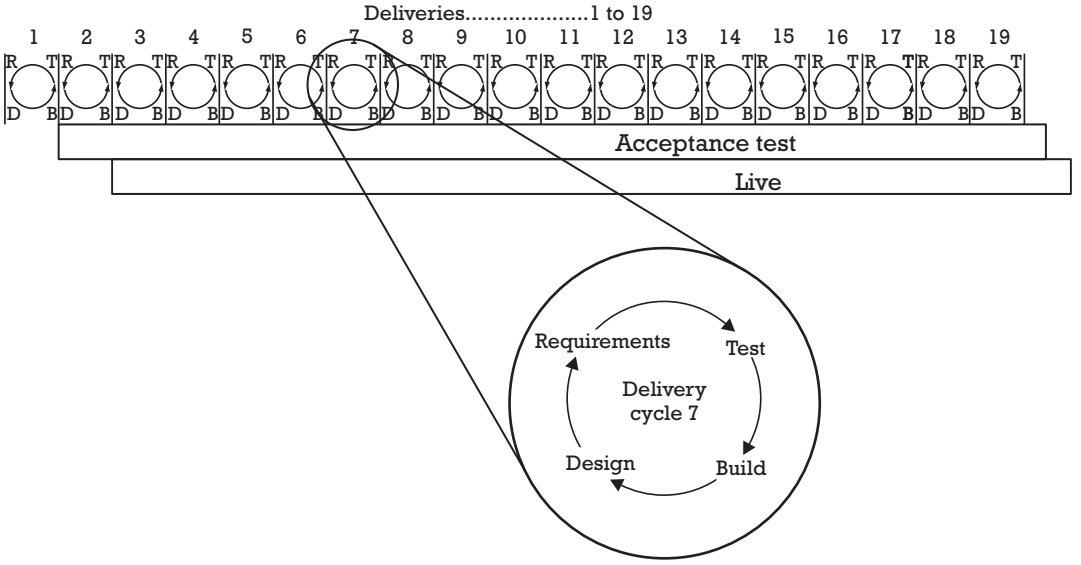


Figure 10.8 Evolutionary model in incremental SDLC.

to live in. They look at and comment on the plans and decide what checks they will do during building. At each step in the early part of the SDLC, products are built that relate to the dynamic stages of testing. Therefore, three things can happen together, as shown in Figure 10.9. Different V- and W-models may show these three steps in slightly different ways but, in essence, what is happening is that:

- A product is built (for example, the requirements, the designs, the code).
- The product is reviewed, by peer review, inspection or walkthrough, to show it matches the products at the previous stage, to show it meets a particular standard, and to show that it can be used as input to the next step.
- Test designs are built that will be used later to run dynamic tests that apply to this product and that demonstrate whether the product is testable.

10.6.5 Advantages and disadvantages of the models

The models each have advantages and disadvantages. These were summarized in Table 10.8.

10.7 Quality views and the models—why we might wish to combine models

We can see in Table 10.11 that the models favor particular quality views, because of which groups are involved in the SDLC.

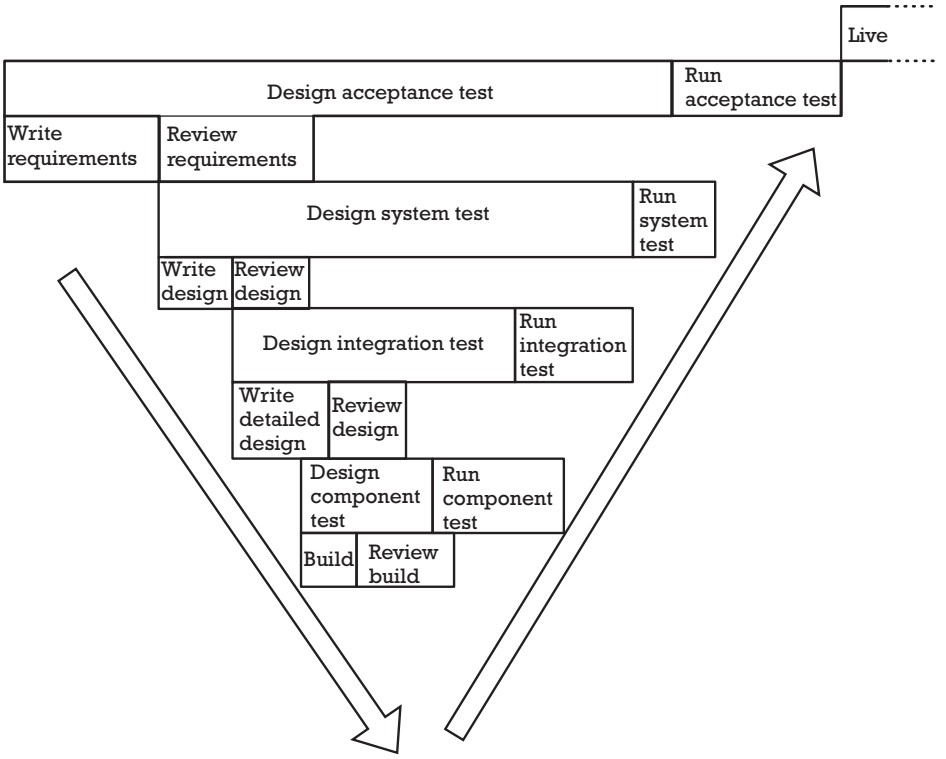


Figure 10.9 V-model.

The models I have shown are generic models. You will find off-the-shelf models and variations described in different companies' standards as well as in books and methodologies (see Table 10.10). You will also see variations on the themes in the models. It is perfectly reasonable to tailor the models to fit what your organization needs, and use published models in a "pick and mix" fashion. However, I do recommend that having looked at the models you settle on a tailored model and a few variations for your organization, with suitable additional tailoring rules. No model is perfect; you will need to use common sense and experience to tailor the model to specific projects, but if you allow a complete free-for-all, chaos will quickly ensue. I have seen situations in which each project is using its own terminology, methods, and measurements, resenting any "outside" ideas. The outcome is that

Table 10.11 Quality Views and the Models

	Life Cycle			
Quality View	*Waterfall*	*Spiral/Iterative/ Evolutionary*	*V-Model*	*Combined*
Transcendent	✓	✓	✓	✓
User view		✓		✓
Value view		✓		✓
Product view		✓		✓
Manufacturing view	✓		✓	✓

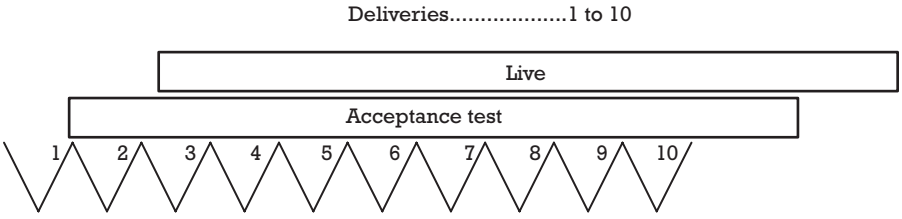
projects cannot be compared, people cannot easily transfer between projects, hostility between projects emerges, and organizations fail to learn from experience because projects guard their own outcomes.

Let us try an example. We look at our organization and assess that:

- Involvement of real customers will help us get the requirements right, and acknowledgment of changing requirements means we want to involve the customer right through the SDLC.
- Appropriate level of control and processes—we do not have very experienced people, but time to market is important so we need to have control without too much bureaucracy.
- Early testing to focus on areas of importance will help us deliver against risks. Maybe static testing (static analysis and reviews) might help us.

From this, we might choose to pick aspects of evolutionary delivery to get important functionality to the customers early, but combine it with some aspects of XP. Because our team is not mature, we would not use the whole of XP, but we might adopt the principle of continuous retest by the programmers and continuous customer involvement. From the V-model, we could use early reviews of requirements and designs, early test design, and static analysis on code. This would be useful if we could involve one of our experienced testers to check the requirements and design for testability, plus to set some detailed entry and exit criteria for steps in which we know we have particular technical weaknesses. To reduce time to market, we might take the risk of combining test levels or removing the formal entry and exit criteria except for high-risk steps. We now have a life cycle like the one in Figure 10.10, tailored to the risks and constraints for our situation [29].

I have seen tailored SDLCs introduced with a marked positive effect, not just on manufacturing quality, but across the quality definitions and groups. For example, in one organization, a framework with entry and exit criteria for the SDLC and for steps within the SDLC gave project teams the ability to tailor for large and small projects. They could split or combine steps and deliverables. The control of the SDLC was through authorization on the



Evolutionary delivery gives us frequent deliveries to live
V-model gives us reinstatement of test levels lost from XP, plus entry and exit criteria at key points

Figure 10.10 Evolutionary, XP, and V-model combined.

entry and exit criteria at each mandatory quality gate within the SDLC. An example of this is shown in Table 10.12, showing an SDLC tailored to a one-page form that describes a particular project. The customer has made a request for what looks like a fairly small and simple piece of work. With the manager and a senior builder, the checklist in Table 10.5 has been completed, and an assessment of the likely cost, size, and risk has been made, to give the Cost · Size · Risk calculation of less than nine. However, the feasibility of providing a simple enough solution is not clear, so a waterfall model would not be suitable. The team believes it will be necessary to produce a prototype to demonstrate that the proposed solution is possible within the constraints of time and budget. A small team of relatively experienced people is available. This suggests that an SDLC tailored to combine some steps is possible, especially as some of the team members are highly experienced in this application area. The team involved will consist of:

- Tom Customer (TC), who has made the request for work to be done, will be involved throughout to help in defining requirements, test prototypes, and test delivered software (user-acceptance test or UAT).
- Mary Builder (MB) (a senior software engineer) and Fred Builder (FB) (a programmer in training) will help TC define requirements, design the solution, build prototypes and deliverables, and carry out their own testing during each iteration.

Table 10.12 Tailored SDLC for the XYZ Project

Iteration and Step	*People During Step, Delivering*	*Exit Review Team*	*Authorize to Move On*
Iteration 1 planning	TC, DM—Plan for iteration 1, based on entry checklist, C · S · R 9	TC, MB, JS, DM, JM, FM	TC
Requirements/ design/build/test	TC, MB, FB—build and test prototype	TC, MB, FB, JS, JM	TC MB
Iteration 2 planning	TC, DM, MB—Refined plan for iteration 2, based on prototype, test results, actual effort, and cost *or* no-go decision	TC, MB, JS, DM, JM, FM	TC, DM
Requirements/ design/build/ initial test	TC, MB, FB—Refine requirements and interface design, first build of solution. JM builds tests, environment for independent test.	TC, MB, FB, JS, JM, FM	TC, MB
Independent test	JM—perform independent test of first build	TC, JM, MB	TC, DM
Iteration 3 planning	TC, DM, MB, JM—Refined plan for iteration 3, based on test results, actual effort, and cost *or* no-go decision	TC, MB, JS, DM, JM, FM	TC,DM
Requirements/ design	TC, MB, FB—Refined requirements and interface design, TC, JS, JM build tests, two environments for independent tests (UAT and OAT)	TC, MB, JS, DM, JM, FM	TC, MB, JS
Build/test	MB, FB—build of solution JM—Independent test of build in environment 1	MB, FB, JM, TC, JS	TC, JS, JM
Independent test	TC, JS—Operational and user-acceptance test (OAT and UAT) in like-live environment 2 leading to decision: Solution ready for delivery to live use *or* no-go decision	TC, JS, JM, MB, FB, DM	TC, JS

- Jenny Supporter (JS) will review the design at each iteration for implications around the infrastructure and existing systems, and will run an operational acceptance test (OAT) as part of iteration 3.
- Dave Manager (DM) will aid with planning and decision making, check that the work is staying within time/budget constraints, and be an escalation point for all team members.
- Joe Measurer (JM) is a specialist tester who will carry out reviews of the requirements and design at each stage, review test results at each iteration, and plan and carry out independent testing in iterations 2 and 3.
- Fiona Measurer (FM) is a quality and process specialist who will review that the SDLC tailoring and the plans are suitable for the level of risk, cost, and size of project.

The team decides to tailor the life cycle. They want to combine the flexibility, replanning, and prototyping of the spiral model, the idea of early tester involvement and reviewing from the V-model, and a three-iteration SDLC. Some of the activities will not have formal exit criteria; effectively they are combined into single steps. The entry criterion for each step is the completion of the previous step plus any preparation activities. Exit reviews and authorization to move on are done formally at significant points given the size of the project.

10.8 Exit from the SDLC

10.8.1 Exit criteria following a detailed acceptance test

If the SDLC completes after an acceptance test based on SMART acceptance criteria like those defined in Chapter 9, it is easy to make the go/no-go decision for taking the software to live use. Either the acceptance criteria have been met or they have not. In Table 10.13, we see simplified example exit criteria for different projects. Notice that the exit from the SDLC may be to deliver software for use, or the SDLC may be stopped and nothing delivered for use. The acceptance test, by which the customer and the supporter decide whether they are prepared to run the software for real, live use may have several outcomes:

- The test passes completely; no problems are identified during the test.
- Some tests have failed but the software is sufficiently fit for purpose that it can be accepted and used with workarounds planned for identified problems.
- The acceptance test fails and the software needs changing; these mean the software will have to be tested again.
- The acceptance test fails and it is judged that the cost and risk of fixing the problems is too great. Nothing is delivered for live use; fixing software is difficult, and prone to cause chain-reaction problems. The SDLC is stopped.

Table 10.13 Example of Exit Criteria for SDLC Stage

Example of Exit Criteria	*Example of Deliverables Documented, Reviewed, and Agreed Upon*	*Example of Authorization Owners*
Acceptance test completed and passed in all aspects; software will move to delivery.	Acceptance test summary report, including acceptance of the implementation/rollback plan, infrastructure, training, documentation, processes, and software.	Process owner, support manager
Acceptance test completed and failed; users and supporters willing to accept as is with documented work rounds.	Acceptance test summary report including acceptance or otherwise of the implementation/rollback plan, infrastructure, training, documentation, processes, and software. Cost–benefit analysis Known problem and workaround list.	Process owner, support manager
Acceptance test completed and failed; rework software and rerun tests.	Acceptance test summary report including acceptance or otherwise of the implementation/rollback plan, infrastructure, training, documentation, processes, and software. Cost–benefit analysis. Rework and rerun plan.	Process owner
Acceptance test completed and failed; cancel project.	Acceptance test summary report including acceptance or otherwise of the implementation/rollback plan, infrastructure, training, documentation, processes, and software. Cost–benefit analysis. Definition of alternative solution.	Process owner

10.8.2 When no exit criteria have been defined

If you have not defined exit criteria from the SDLC, how will you know when to stop? It might be that you make the decision to go live because you have run out of time or budget, or because you have completed all the planned tests, or because our manager will get angry if we do not deliver soon. Any of these could be the right or the wrong reason for stopping. If we think of the quality definitions and the groups, each group will have a preferred quality reason for stopping or continuing:

- Customers may be desperate for the new software, and they will go with fit-for-purpose rather than perfect, or, indeed, what is in the specification, but may be less sanguine once defective code is delivered.
- Managers will be watching the time and cost of the SDLC—they want to ensure the organization gets a proper return on investment and does not spend excessively.
- Builders want to ensure that they have implemented all the features and attributes from the specification.
- Measurers can become excessively focused on defect identification and removal at the expense of time and budget constraints.
- Supporters will go with fit-for-purpose rather than perfect, or, indeed, what is in the specification, but may be less sanguine once defective code is delivered.

If no exit criteria are defined and agreed on between the groups, the differing viewpoints will mean that the team argues about whether the software is ready for delivery or not. If you are approaching the end of an SDLC without agreed exit criteria, then I suggest you attempt to document and agree on some criteria for acceptance (see Section 10.8.5). You also need to identify the circumstances under which the software would not be accepted and the actions to be taken in that case.

To prevent this problem from recurring, improve the SDLC model that is used in your organization by adding exit criteria to the model.

10.8.3 When exit criteria have not been met

As with the entry criteria, if the exit criteria have not been met, we must not exit the SDLC. In order that we can be strict, it is important that the exit criteria are:

- Objective;
- Defined in a way that is SMART;
- Necessary rather than simply desirable.

However, there are occasions when we might decide to waive the exit criteria. This may happen if we decide that the exit criteria are excessive or incorrect, or if we believe this is an emergency. There are also occasions when we may wish to increase the exit criteria, for example, if we see that the SDLC is at higher risk than our normal projects. Exactly the same lessons apply as we discussed in Section 10.3.3, so I will not repeat them here. It suffices to say that the exit from the SDLC should match to the entry to delivery (Chapter 11).

10.8.4 Tailoring exit criteria

Exit criteria will be similar for different SDLC models, but there may be differences for specific projects or pieces of work. As with the entry criteria, tailoring against cost, budget, and risk measures may help teams to choose appropriate exit criteria for a particular project. Table 10.14 shows some examples.

10.8.5 When no acceptance criteria have been set

However late it is in the SDLC, if we do not have acceptance criteria, we should try to set some. For the acceptance criteria, it may well be too late to do anything about attributes the team has not considered. Use Table 9.8 in Chapter 9 to help you build a checklist of attributes to consider as acceptance criteria.

To prevent this problem from occurring in the future, improve the process to include a detailed start-up for the SDLC.

Table 10.14 Example of Exit Criteria Tailored to Projects

	Example of Exit Criteria	
C · S · R Score	*Example of Exit Documents*	*Example of Authorization Owners*
1–5	1. E-mail with notification of change completion and request to move code to live from builder to supporter, and customer, copied to manager and measurer Deliverables: E-mail and acknowledgment, code standards checklist complete by builder, customer test results	Customer, supporter
6–12	1. As above, plus 2. Request raised at monthly change management meeting 3. Tests completed by independent tester and by customer/supporter Deliverables: As above, plus acceptance criteria completion report, evidence from testing. and known fault list	Customer, process owner, supporter builder, manager
12–20	1. As above, plus 2. Acceptance criteria metrics from ISO 9126 completed and reviewed 3. Separate unit, system and acceptance test stages complete Deliverables: As above, plus acceptance criteria documentation, evidence from all testing stages and known fault/workaround list	Customer, process owner, project manager
20–27	1. As above, plus 2. Additional risk assessment on delivery carried out by process owner and project manager 3. Implementation planning, including contingency planning completed 4. Project authorization board has discussed and agreed to the implementation and risk plan for delivery Deliverables: As above, plus additional work on risk assessment, achievement of aims and objectives, targets and indicators, constraints, acceptance criteria, implementation plan, project authorization to complete	Project authorization board, SDLC project sponsor, process owner, project manager

10.9 Conclusion

No single SDLC model is perfect. What we choose depends on our definition of quality, our constraints and our risks. All the life cycles have similar steps, and we must tailor SDLC models to fit the risk level and constraints within a particular project.

There are more deliverables that just the software systems. We must also plan for delivery of:

- Infrastructure, operational, and support processes and equipment;
- Training (for software, support, and business processes);
- Documentation (for software, support, and business processes);

- Business and manual processes;
- Implementation processes, including rollout and rollback plans;
- Data, for example, during a data migration, both databases and standing data.

It is worth having some pretailored SDLC models, associated with different levels of risk and different types of projects. All SDLCs need some start-up beforehand; you cannot be successful with any SDLC unless you have fulfilled the entry criteria. All SDLCs require good communication between all groups. All are aimed at delivery, which we will look at in Chapter 11.

References

[1] Reid, S. C., "Software Testing Standards—Do They Know What They Are Talking About?" http://www.testingstandards.co.uk/publications.htm, August 2003.

[2] Software Engineering Institute, "Capability Maturity Model®," http://www.sei.cmu.edu, July 2003.

[3] Humphrey, W., *Introduction to the Team Software Process,* Reading, MA: SEI, 2000.

[4] IT Infrastructure Library, *Best Practice for Service Management*, London, England: Office of Government Commerce, 2002.

[5] Nance, R. E., and J. D. Arthur, *Managing Software Quality*, New York: Springer-Verlag, 2002.

[6] Boehm, B. W., "A Spiral Model of Software Development and Enhancement," *IEEE Computer,* Vol. 21, No. 5, 1988, pp. 61–72.

[7] Kruchten, P., *The Rational Unified Process,* Reading, MA: Addison-Wesley, 1999.

[8] Beck, K., *Extreme Programming Explained,* Reading, MA: Addison-Wesley, 2001.

[9] DSDM, DSDM, http://www.dsdm.org, April 2003.

[10] Gilb, T., papers on Evolutionary Delivery on http://www.gilb.com, including "Competitive Engineering: A Handbook for Systems and Software Engineering," http://www.gilb.com, September 2003.

[11] Craig, R. D. and S. P. Jaskiel, *Systematic Software Testing,* Norwood, MA: Artech House, 2002.

[12] Gerrard, P., and N. Thompson, *Risk Based E-Business Testing,* Norwood, MA: Artech House, 2002.

[13] BSI, BS7925-2:1998, Software Testing—Part 2: Software Component Testing, BSI 1998.

[14] Belbin Associates, "Belbin Team Roles," http://www.belbin.com/belbin-team-roles.htm, October 2003.

[15] Team Technology Web site, "Working Out Your Myers Briggs Type," http://www.teamtechnology.co.uk/tt/t-articl/mb-simpl.htm, October 2003.

[16] Honey, P., "Learning Styles," http://www.peterhoney.co.uk/product/ learning styles, October 2003.

[17] Kirton, "Adaptors and Innovators Defined," KAI Web site, http://www.kaicentre.com/, July 2003.

[18] de Bono, E., *Six Thinking Hats®*, New York: Penguin, 1999.

[19] Robson, M., *Problem Solving in Groups,* Aldershot, England: Gower, 1995.

[20] TQMI, *Problem Solving—Tools and Techniques,* Frodsham, England: TQMI, 2001.

[21] IEEE 1028™ Standard for Software Reviews, 1997.

[22] Gilb, T., and D. Graham, *Software Inspection,* Reading, MA: Addison-Wesley, 1993.

[23] Hohmann, L., "Lo-Fi GUI Design," *Software Testing and Quality Engineering,* Vol. 1, No. 5, September 1999.

[24] "Drawing Concerns: A Structured Rich Picturing Approach," http://business.unisa.edu.au/cobar/documents/richpic_colin.pdf, November 2003.

[25] Rose, J., "Soft Systems Methodology as a Social Science Research Tool," http://www.cs.auc.dk/~jeremy/pdf%20files/SSM.pdf.

[26] Buzan, T., *The Mind Map Book,* London, England: BBC Consumer Publishing, 2003.

[27] Freeburn, G., "Mind Mapping 101 for Testers," *EuroSTAR Conference,* Edinburgh, Scotland, 2002.

[28] Pavyer, E., "An Introduction to Earned Value Management," *Project Manager Today,* Vol. 11, April 2003.

[29] Evans, I., "The Risks We Take with Testing," *British Quality Foundation IT&T Group Meeting,* February 2003.

Selected bibliography

Belbin, R. M., *Management Teams—Why They Succeed or Fail,* London, England: Butterworth Heinemann, 1981.

Belbin, R. M., *Team Roles at Work,* London, England: Butterworth Heinemann, 1995.

Gilb, T., *Principles of Software Engineering Management,* Reading, MA: Addison-Wesley, 1988.

Gnatz, M., et al., "The Living Software Quality Process," *Software Quality Professional,* June 2003, pp. 4–16.

Honey, P., and A. Mumford, *The Learning Styles Helper's Guide,* Maidenhead, England: Peter Honey Publications, 2002. PeterHoney.com, 10 Linden Avenue, Maidenhead, Berks, SL6 6HB. Tel.: 0162863946. Fax: 01628633262. E-mail: info@peterhoney.com.

Kroeger, O., J. M. Thuesen, and H. Rutledge, *Type Talk at Work: How the 16 Personality Types Determine Your Success on the Job,* New York: Bantam Doubleday Dell, 2002.

McHale, J., "Innovators Rule OK—Or Do They?" *Training & Development,* October 1986, http://www.kaicentre.com/.

Sticky Minds Web site Roundtable, facilitator Craig, R., "What Is Software Quality and How Do You Measure Its Value?" http://www.stickyminds.com/s.asp?F=S6540_ROUND_46, August 2003.

CHAPTER

11

Delivery and Support When Going Live

Contents

In this chapter I shall:

- Describe the entry criteria, steps, and exit criteria for delivery;
- Identify the quality definitions from Chapter 1 that prevail at this stage;
- Identify how each of the groups identified in Chapter 2 become involved at this stage;
- Introduce techniques that aid teamwork at this stage.

It's 10 p.m. on Sunday; either we complete the implementation and work around the defects, or we roll back now. We cannot wait any longer, or there isn't time to roll back and be ready for Monday.

—Customer makes an assessment of the path of least damage during a flawed implementation

11.1 Delivery—description

11.1.1 Delivery considerations

Delivery is the point in the life span of software after the software-development life cycle (SDLC) has completed when products are moved to live use. It is a (relatively) short but anxious period of time, but it is highly importance; many activities will come together at a single milestone in the project plan. Delivery is not always a success; sometimes the implementation fails and we have to roll back by removing the software and restoring the systems to their predelivery state. "Postponement of a system's implementation is painful

and often very costly. The implementation of a poor system, however, is much more costly and also much more painful" [1]. The roots of a failed delivery are found in the entry and exit criteria for the SDLC, which I discussed in Chapters 9 and 10.

Delivery comes between the SDLC and the postdelivery period when the software is used (Figure 11.1). As with the earlier stages, all the groups have a significant contribution to make but they do not always realize their part. In this chapter, I will discuss the activities carried out during delivery, and how all the team can contribute to making it a success. Specific activities need to be planned and managed leading to the delivery point if the delivery is to be achieved without problems.

There are things to consider when planning the delivery, depending on the type of software, the other deliverables that come with the software solution, and the type of customer:

- *Installation team:* The installation of the software plus any other products may be done by the customer, or by a team from the support group (operations or help desk) or by the build team themselves.
- *Method of delivery:* Delivery and installation of the software might be by different media (download from Internet/intranet, by transfer within a configuration management system, by use of removable media such as diskettes, tapes, or CDs). Delivery of other products such as training and support may be face to face, via documentation, or via other media such as video and CD.
- *Time period for installation:* Delivery and installation may be a short process, taking minutes, or it may be an extended process. In the latter case, it may take place overnight, over a weekend, or over a longer period. For a large installation, it may be phased in geographically or in some other way, or there may be a pilot project.
- *Urgency of the delivery:* Sometimes, emergency changes to systems are required, and in this case the change may require a special process in order to speed up the delivery without losing control.
- *Support around the delivery:* A small delivery may not require particular support; perhaps the minimum would be a release notice with a known problem list. Larger deliveries may be supported by training, which

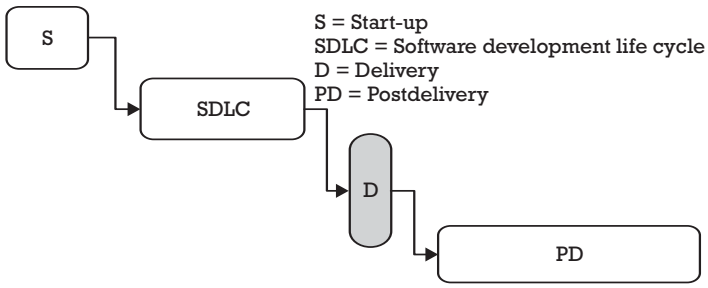


Figure 11.1 Life span stages.

would then be a deliverable in itself. Other support to a software delivery might include expert teams to work on support lines or with the software users. The need for additional support depends on how transparent the delivery is to the people using the IT systems.

- *Quality measures of the deliverables:* The release notice will typically list any known problems and workarounds for the deliverables. The quality measures for the deliverables will affect the decision to make the delivery or not. This includes an assessment of the known problems and whether the exit criteria for the SDLC were met.
- *Size of delivery:* Different delivery strategies may be required for new "green field" systems, small and large changes to existing systems, single changes, and groups of changes.

11.1.2 Identifying the delivery

A delivery will consist of one or more of the following deliverables, built and tested during the SDLC:

- Software;
- Infrastructure, for example new hardware, communications equipment, systems software, plus operational and support processes;
- Training material and delivery of training for software, support, and business processes;
- Documentation for software, support, and business processes, for example, user and support guides;
- Business and other processes as well as process changes, for both manual and IT processes;
- Implementation processes, including rollout and rollback plans;
- Data, for example, during a data migration, both databases and standing data.

The delivery plan should identify all the deliverables, each identified with its version number and status, under a configuration management system. A configuration management system is used to make sure we deliver the right version of each deliverable; this is because as we build and test deliverables, they will go through many revisi ons. It is an easy human error to deliver the wrong version. The group of deliverables to be released together may be identified with a release ID [2]. This is updated for later deliveries of changes to an existing system so that the size and type of the delivery is identified in the release ID: for example, if the first release into live use is identified as "V1":

- "V2" is a major release that contains significant changes to V1.
- "V1.1" is a minor release of small changes to V1.
- "V1.1.1" is an emergency fix to the system.

In an SDLC that takes place as part of an organization or business transfer, the point of delivery might also be the point at which the business is responsible for transferred customers and staff. Although this would not normally be considered as part of the SDLC delivery, the introduction of these people to the new organization would need to be managed as part of an overall business delivery process, including introductions and induction training.

11.2 Delivery viewpoints

Each group has a view of when the point of delivery happens, and of what is included in the delivery. For the builders, they may only be interested in the deliverables for which they have direct responsibility. For example, within the build group, developers will be interested in the delivery of their code, trainers will be interested in the delivery of the training, and technical authors in the delivery of the user guides. It is interesting to note that in *Introduction to the Team Software Process* [3], the final step in the software engineering project is the *postmortem,* which comes immediately after the system test; the implication is that the software engineers hand over the code at this point, before the acceptance test, so the user-acceptance test would be carried out by the customers, leading into the delivery, to be carried out by customers and supporters. From the viewpoint of the supporters, the delivery covers code and the infrastructure to support it. For example, in the IT Infrastructure Library (ITIL) manual *Best Practice for Service Support* [2] under Release Management, the main components of a release for delivery are identified as:

- Application programs, developed in-house or externally, including packages;
- Utility software;
- Systems software;
- Hardware and hardware specifications;
- Assembly instructions and documentation, including user manuals.

Notice how this includes everything in which the supporters will be interested, but not training or business processes, which some IT people would regard as outside the SDLC. Yet, from the customer viewpoint, without the business processes and training, it may not be possible to take delivery of the software successfully.

Thus, we see that if we take a simple view of the point of delivery, we may forget to deliver something needed by one of the groups. Some projects may see the delivery as being the responsibility of the customers and the supporters, with the managers, builders, and measurers simply handing over tested code and no other material. The managers, builders, and measurers have taken the system to the end of either system test for handover to

the acceptance test, or have supported it to the end of the acceptance test. At acceptance, the project is complete. The deliverables are handed over, and the customer and supporter proceed on their own. The supporter perhaps carries out the installation of the software, and the customer prepares to use it, perhaps by reading the user guide. In other projects, such as the delivery of a commercial off-the-shelf (COTS) package, the customer may not even have a supporter to carry out the installation. For example, in a one- or two-person company, the acquisition of a desktop computer, with word processing and accounting software, may only require a purchase order and a visit to the local computer store, with the person using the PC carrying out the entire installation themselves.

As we see in Table 11.1, if only the customer and supporter are involved, this results in a limited quality viewpoint, which may or may not matter depending on the size and complexity of the delivery.

If the SDLC has been carried out with a viewpoint limited to the builder, measurer, and manager (see Table 10.1 in Chapter 10) we may quickly find that the customer and supporter views of whether they have received a quality system is at odds with the others' views of whether they have delivered quality.

In many large projects, particularly for custom-made or tailored systems, the builders, measurers, and managers will aid in the delivery, and may well have a team to support the customers and supporters as the customers take delivery of the software and start to use it. Although members of all groups will support the delivery, their viewpoints will be quite different (Table 11.2):

Table 11.1 Views of Quality at Delivery—Simple

	Groups	
Quality View	*Customer*	*Supporter*
Transcendent view	✓	✓
User view	✓	✓
Value view		
Product view		✓
Manufacturing view		

Table 11.2 Views of Quality at Delivery—Teamwork

	Groups				
Quality View	*Customer*	*Manager*	*Builder*	*Measurer*	*Supporter*
Transcendent view	✓	✓	✓	✓	✓
User view	✓			(✓)	✓
Value view		✓			
Product view			✓	✓	✓
Manufacturing view			✓	✓	

✓ is a primary quality view.
(✓) is a quality view that may be taken by some people in this group.

- Customers are very engaged at this point. Will the system work for real? Will they be able to use it? Will the delivery go well? From the user quality viewpoint, will the system be fit for purpose?
- Supporters are also very engaged; they will be rolling out the system and starting to support it. Their questions and viewpoint are similar to that of the customer, with the addition of a product viewpoint that looks at the effect of attributes in the system as a whole; for example, how does the new software affect performance of the overall system?
- Managers have quite a different reason for anxiety at this point. Will the project team be able to finish this project and (with more or less relief) move onto a shiny new project? Using the value quality viewpoint, did the project come in within budget and time scale, and can any delays in delivery be prevented?
- Builders will be pleased to have completed the project; they will want to see a clean sign-off against the delivery. From their manufacturing and product quality views, is the delivery defect free and does the software have all the specified attributes?
- Measurers will share the builders' views and, in addition, will be worried about their efforts in assessing quality from the manufacturing view. They will be focused on defects identified during delivery that were not identified earlier.

For a release of any complexity, it is likely that many people will be involved. It may be useful to have a *responsibility matrix* [2] to show who is involved and with what responsibilities for the different types of SDLC and projects. This might be arranged by size and risk, as well as by technical content of the delivery; for example, whether the delivery is of a bought package, a customized package, database changes, hardware changes, or custom-made software. For example, in contrast with our one- or two-person company above, in a large organization, depending on its processes, the purchase and installation of a PC with word processor and accounting software may require involvement from a logistics group, the desktop support group, the change management team, IT security, an electrical equipment safety team, and so on. The authority and responsibilities for delivery tasks would be included in the responsibility matrix, which could be incorporated with the entry and exit criteria for delivery.

In particular, the support of builders and measurers will be needed to aid in resolving any problems that arise during the delivery. Managers need to be involved to make cost–benefit analysis for any change of plan during the delivery.

The period around the delivery needs to be managed carefully because change is taking place. The unfamiliar system requires a learning curve from the customers and supporters. There also may well be "teething problems" as the system is first used. The enthusiasm for the new software may

drop significantly if benefits from the delivery cannot be realized right away. Enthusiasm for the system may not be regained until significant benefits have been demonstrated for the system. This is referred to as the "Silver Bullet Life Cycle" [2]. Managing the relationships and communication between the groups is vital in order that the team works through the delivery taking account of changes in attitude and confidence as well as the technical activities. Using the teamwork techniques listed in Table 11.3 and described in Appendix A will help to manage the communications.

11.3 Entry criteria for delivery

The entry criteria for delivery must include all the exit criteria for the SDLC, but may include additional tasks required for the delivery into live use, if these tasks have not been defined as part of the SDLC. Table 11.4 shows some example entry criteria for which preparation for delivery in terms of an implementation and rollback plan has been completed within the SDLC, but other aspects of infrastructure and service support have not been included.

As the SDLC may not include consideration of release and rollout plans, the entry criteria for delivery should explicitly mention completion of tasks beyond simply the delivery of the software. The specific entry criteria will depend on the type of organization and the type of delivery. In particular, evidence of acceptance testing of a delivery should include testing not just the functionality of the software, but ascertaining that it can be installed and backed out (deinstalled) [2].

Apart from use of the software, there are a number of services that are needed postdelivery, and I will discuss them in Chapter 12. We should not need to carry out detailed planning and delivery of these services for every SDLC; these activities should be in place in order that we can support software systems. For a project in which we are acquiring a new system on a new infrastructure, we would plan and set them up as part of the whole IT infrastructure, so our entry criteria would include checking that all the support and infrastructure is in place to support the live system. For a project in which we are delivering new software to an existing infrastructure, we would need to review whether changes were needed in any of these areas to support the delivery and use of the software. The completion of these changes would become part of the entry criteria for delivery for that particular project. A very good description of the type of infrastructure activities required and checklists for reviewing delivery plans are in the ITIL Service Delivery and Service Support Manuals [2, 4].

11.4 Delivery—typical activities

The activities that take place during delivery depend on the size and complexity of the project. This is best illustrated by some examples.

Table 11.3 Summary of Techniques for Teamwork in Delivery

Subject Area	*Technique Examples*	*In 30 seconds… (see Appendix A for more)*
Delivery	Pilot projects [5, 6]	A limited delivery to a small area, which enables the customers and suppliers to test delivery and use of the proposed system before rollout to a large number of sites.
Delivery	Release management [2, 4, 7]	"Release management builds the final customer solution, which also includes instantiation of the developing and testing environment. In a product-line context, release management includes the release of core assets to product developers. When shopping for CM tools, make sure the one you buy can help you build releases." [7]
Delivery	Build management [2, 4, 7]	"Build management enables developers to create a version of a product, which can be anything from a single component to a complete customer solution for the purpose of testing and/or integration." [7]
Team relationships and natural roles/team skills	Belbin team roles [8]	The SDLC can fail because of personal rather than technical factors. Teams need to understand their strengths and weaknesses as a team. A balance of roles/skills is required in the personalities in the team. Example roles: plants have new ideas; completer–finishers want to finish to fine detail. Too many plants and you will never finish anything.
Improve communication—empathy with others	MBTI [9]	Different people have different personalities and communication styles. People who wave their arms around and talk a lot can annoy people who like to be quiet and think, and vice versa. The Myers-Briggs Type Indicator (MBTI) identifies four contrasting type pairs (e.g., *Introvert/Extrovert*) leading to 16 "types" (e.g., INTJ is *Introvert-iNtuitive-Thinking-Judging*).
	Honey & Mumford [10]	The Honey & Mumford Learning Styles Questionnaire identifies preferred learning styles (e.g., *Pragmatists vs. Theorists*) require different experiences to learn.
	Kirton [11]	Kirton identifies preferred problem solving methods (*Adaptors vs. Innovators*)—do we break the rules or work within them?
Improve meetings	De Bono's Six Thinking Hats [12]	Improve meetings by setting rules for behavior. Six "hats" are used. Everyone wears the same color hat at the same time: Blue Hat—meeting structure, Black Hat—pessimistic, Yellow Hat—optimistic, Red Hat—feelings, White Hat—facts, Green Hat—creative ideas. Allows meeting members to move outside their stereotypes and allows time for different, sometimes difficult types of communication.
Identify problems and root causes, find solutions	Ishikawa fishbones [13, 14]	Use to identify problems, root causes of problems and solutions. On a fishbone diagram, brainstorm problems, their possible causes, their root causes, and, therefore, solutions to the root cause.
Review documents and other products	Reviews [15, 16]	There are five types of review: management review, technical review, inspection, walkthrough, and audit—all of which are relevant throughout the SDLC. Specialist testers regard them as a form of testing, because they are used to find and prevent defects in products and processes.
Understand whether an idea is worth pursuing	Risk workshop by brainstorming [13, 14]	A brainstorming workshop is run to list all the possible risks people can identify. The risks are sorted into groups, separating risks (might happen in the future) from issues (problems right now) and constraints. The risks are scored for impact and likelihood, and ranked to give a prioritized list.
	Cost–benefit analysis [13, 14]	Cost–benefit analysis is done by calculating the predicted benefits of the proposed change (time, money saved) and setting this against the predicted cost.

Table 11.4 Example Entry Criteria for Delivery

Example of Entry Criteria	*Example of Entry Documents*	*Example of Authorization Owners*
1. Acceptance test completed and passed in all aspects; software will move to delivery	Acceptance test summary report, *including acceptance of the implementation/rollback plan, infrastructure, training, documentation, processes, and software*	Process owner, support manager
2. Acceptance test completed and failed; users and supporters willing to accept as is with documented workarounds	Acceptance test summary report, *including acceptance or otherwise of the implementation/rollback plan, infrastructure, training, documentation, processes, and software* Cost–benefit analysis Known problem and workaround list.	Process owner, support manager
3. Hardware change: Either 1 or 2 completed, plus hardware delivery complete	Hardware implementation test complete and accepted	Infrastructure manager
4. Additional support line/help desk required: Either 1 or 2 completed, plus support plan implemented	Support policies and processes complete and agreed on Support line/help-desk infrastructure (hardware, communications, and software tools) complete and accepted Support personnel trained and ready	Support line/ help-desk manager
5. Change to business processes required: Either 1 or 2 complete plus business process changes agreed on, tested and accepted, and business training complete	Business processes agreed on Business personnel trained and ready	Process manager, customer services manager

11.4.1 Person buys PC and software for self-installation

The builders, measurers, managers, and supporters will work for the PC and software vendors. They must prepare the product with everything the customers will need to support them through delivery and into use of the system. The level of intervention by a specialist with technical knowledge of PCs will need to be reduced by improving the installation process itself, perhaps by automation, but also by providing step-by-step instructions not just for the installation of the software but including steps to take before and after the installation. Additionally, supporting documentation should be written in the customer's terminology, not in IT terminology. The delivery activities that might be typical here would be for the customer to:

- Check all the delivered components against a supplied parts checklist.
- Assemble hardware following a step-by-step guide
- Boot up system and follow on-screen initialization steps.
- Customize system.
- Back up system.
- Install purchased software package, following steps in the installation guide.

- Check that the installed software works as expected.
- If it does work, make a backup of the system.
- If it does not work, deinstall following the manufacturer's deinstallation instructions.

Of course, at the point where the customer finds that the software does not work, they may want to take a number of actions: following the troubleshooting guide supplied by the manufacturer, ringing the help line, looking at on-line help, or asking for their money back [17].

11.4.2 Single-site delivery of software

For a delivery of new software or an upgrade to an existing system, the steps within delivery need to protect processes or people using the system from unexpected changes. The delivery steps might include:

- Awareness, education, and training prior to the delivery—people are made aware that a change will be made and given any training they will need, plus details of additional support such as additional helpline numbers.
- Carry out normal day's processing on system.
- Carry out end-of-day processes on system.
- Back up system.
- Overnight, install new software, following steps in the installation procedures.
- Check that the installed software works as expected.
- If it does work, make a back up of the system and make ready to go live.
- If it does not work, deinstall following the deinstallation instructions and make ready to go live with the old system.

If the installation was not successful, the team may chose to go live with the changes anyway; the exit criteria for the delivery would need to define the acceptance levels.

11.4.3 Multisite rollout of new software to existing infrastructure

The steps will be similar to the single-site delivery but will be repeated at each site, and there may be a delay in the rollout or installation to allow the people using and supporting the system to become used to it. For a multisite installation a phased approach is easier to manage and less prone to error than a "big bang" approach [6]. For new software, for example taking on a software tool that replaces manual activities with a radical change in process, a pilot project may be useful. In fact, pilots are useful when the customer for the delivery is the IT group itself; for example, when taking on a new testing

tool, piloting the use of the tool in a small group and then rolling out to the rest of the IT teams is generally recommended [7]. The steps might include:

- Choose pilot and make a draft implementation rollout plan for remaining sites.
- Awareness sessions for the pilot and release plan.
- Implement on the pilot system, as a single-site process, including education and training.
- Monitor and evaluate the pilot, confirm or update rollout plan.
- If proceeding, roll out to the next site, as a single-site process, including education and training.
- If not proceeding, roll back.
- Monitor and evaluate the site, confirm or update rollout plan.

If the pilot is not successful, a decision needs to be made about how to proceed, so the pilot needs its own acceptance and exit criteria. For a with problems, we might decide to go ahead with the implementation rollout anyway, or to implement it only in certain areas, or to change/rework the software and repilot, or roll back and look for a different solution.

Once the pilot is complete, if a part of the rollout is not successful the next move is more difficult; this means that rollback plans should be considered in detail at each stage. If the pilot and the first phase of rollout have been accepted, but the second phase fails, should we roll back just the second phase or to the first phase and pilot as well? Will it still be possible to roll back if data on the systems relies on the new software? Will formats still be compatible? The implication of rollout and rollback must be considered in the delivery planning.

11.4.4 Data migration project software and hardware changes

In this type of project, the delivery is more complex. The project might require making changes to the target system to ready it for the new data, removing data from the source system, perhaps converting its format, loading it to the target system, and checking that all the data has been moved. The delivery may take place over several days, and activities on different systems may be taking place in parallel:

- Prior to the migration, deliver the software and hardware changes to the target system (see Section 11.4.3).
- Carry out final day's processing on source system.
- Carry out end-of-day processes on source system.
- Back up source system.
- Extract source data.
- Final day's processing on target system.
- Carry out end-of-day processes on target system.

- Back up target system.
- Convert data.
- Load converted data.
- Check that all data has loaded.
- If all data has loaded successfully, go live on target system.
- If data has not loaded successfully, restore to backup on source and target.

If the data migration was not successful, the team may chose to go live anyway; the exit criteria for the migration delivery would need to define the acceptance level for the data. Migrations may be carried out as a big bang or with a phased approach. As with the multisite approach, the delivery plan must consider the implications of rollout and rollback.

11.5 Exit from delivery

Exit from delivery takes place when the exit criteria for delivery have been met. This will either be:

- When the release activities have completed successfully and the system is ready for live use; it will then move into the postdelivery stage (Chapter 12);
- When the release is declared unsuccessful and the release has been rolled back.

Table 11.5 lists some examples of exit criteria for delivery.

11.6 Conclusion

The delivery requires involvement of all groups, whether it results in a successful rollout or in a rollback. There are standard IT infrastructure processes that support the delivery.

Table 11.5 Example of Exit Criteria from Delivery

Example of Exit Criteria	*Example of Deliverables Documented, Reviewed, and Agreed Upon*	*Example of Authorization Owners*
Installation and release complete, ready for live use	Delivery checklist complete and reviewed Updated IT service Updated configuration management system (new software, live version) Decommission checklist complete Known defect list Resolved defect list	Process owner, IT infrastructure manager
Release rollback complete	Deinstallation checklist complete and reviewed System restore to preinstallation back up complete	Process owner, IT infrastructure manager

References

[1] Pol, M., and van E. Veenendaal, *Structured Testing of Information Systems*, Deventer, the Netherlands: Kluwer BedrijfsInformatie, 1998.

[2] IT Infrastructure Library, *Best Practice for Service Support,* London, England: Office of Government Commerce, 2002.

[3] Humphrey, W., *Introduction to the Team Software Process,* Reading, MA: Addison-Wesley, 2000.

[4] IT Infrastructure Library, *Best Practice for Service Delivery,* London: Office of Government Commerce, 2002.

[5] Craig, R. D., and S. P. Jaskiel, *Systematic Software Testing,* Norwood, MA: Artech House, 2002.

[6] Fewster, M., and D. Graham, *Software Test Automation*, Reading, MA: Addison-Wesley, 1999.

[7] Software Engineering Institute, "A Framework for Software Product Line Practice," v4.1, Software Engineering Institute Web site, http://www.sei.cmu.edu.

[8] Belbin Associates, "Belbin Team Roles," http://www.belbin.com/belbin-team-roles.htm, October 2003.

[9] Team Technology Web site, "Working Out Your Myers Briggs Type," http://www.teamtechnology.co.uk/tt/t-articl/mb-simpl.htm, October 2003.

[10] Honey, P., "Learning Styles," http://www.peterhoney.co.uk/product/learning styles, October 2003. PeterHoney.com, 10 Linden Avenue, Maidenhead, Berks, SL6 6HB. Tel.: 01628633946. Fax: 01628633262. E-mail: info@peterhoney.com.

[11] Kirton, M. J., "Adaptors and Innovators Defined," see KAI Web site, http://www. kaicentre.com/, July 2003.

[12] de Bono, E., *Six Thinking Hats®*, New York: Penguin, 1999.

[13] Robson, M., *Problem Solving in Groups*, Aldershot, England: Gower, 1993.

[14] TQMI, *Problem Solving—Tools and Techniques*, Frodsham, England: TQMI, 2001.

[15] IEEE 1028™ Standard for Software Reviews, 1997.

[16] Gilb, T., and D. Graham, *Software Inspection,* Reading, MA: Addison-Wesley, 1993.

[17] Kaner, C., *Bad Software,* New York: Wiley, 1998.

Selected bibliography

Belbin, R. M., *Management Teams—Why They Succeed or Fail*, London, England: Butterworth Heinemann, 1981.

Honey, P., and A. Mumford, *The Learning Styles Helper's Guide,* Maidenhead, England: Peter Honey Publications, 2002. PeterHoney.com, 10 Linden Avenue, Maidenhead, Berks, SL6 6HB. Tel.: 01628633946. Fax: 01628633262. E-mail: info@peterhoney.com.

Kroeger, O., J. M. Thuesen, and H. Rutledge, *Type Talk at Work: How the 16 Personality Types Determine Your Success on the Job,* New York: Bantam Doubleday Dell, 2002.

McHale, J., "Innovators Rule OK—Or Do They?" *Training & Development,* October 1986, http://www.kaicentre.com/.

CHAPTER

12

The Life of a System Postdelivery

Contents

In this chapter I shall:

- Describe the entry criteria, steps, and exit criteria for the postdelivery stage;
- Identify the quality definitions from Chapter 1, which prevail at this stage;
- Identify how each of the groups identified in Chapter 2 become involved at this stage;
- Introduce the types of support activities carried out at this stage, especially for readers who are not support specialists, including the different types of maintenance changes to the delivered system;
- Introduce techniques that aid teamwork at this stage.

My team has been given the assignment of picking up after this latest "wow project." We need to run several manual batch jobs each day and I'm getting beaten up because the support cost has gone up by 30% since implementation. I think the project team got a bonus for this mess!

—Cynical application maintenance manager wondering why no one involves supporters until after delivery!

12.1 Postdelivery—description

Postdelivery covers the life of the software from when it is first delivered and while it is being used until it is decommissioned. It is the longest part of a software system's life span. During this time, customers use the software while supporters maintain and support it. It comes after the delivery stage covered in Chapter 11 (see Figure 12.1).

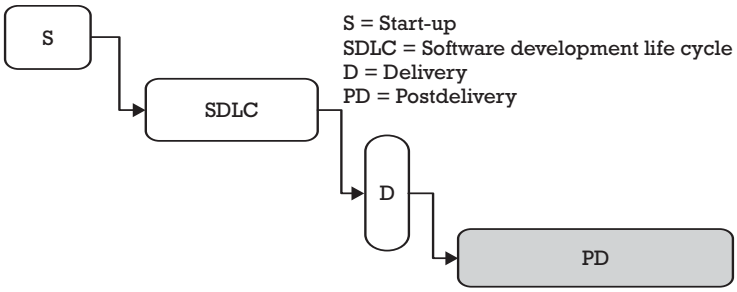


Figure 12.1 Life span stage.

12.1.1 Postdelivery for different types of software acquisitions

Postdelivery activities will be different for different types of customers, systems, and software. Let us look at some examples.

12.1.1.1 Commercial off-the-shelf (COTS) package installed by customer

The customer installs the software and uses it. Postdelivery, if the software does not work as expected, they might deinstall and replace it by something else, contact the vendor's help line for assistance, or find a workaround by themselves. The vendor may advertise *patches* that fix reported bugs, and these may be delivered to the customer on request for the customer to install. If the vendor produces a new version of the software, with enhancements, the customer may choose to install the upgrade or to continue using the existing system without the upgrade. The postdelivery activities are:

- Use of the software;
- Customer applying patches and upgrades when they are ready, if wanted;
- Deinstallation.

12.1.1.2 System developed in-house

The software system is likely to be used during office hours by the user group (customers). There may be overnight batch processing run on behalf of the user group by the operations team (supporters). From time to time maintenance changes will be applied. These may be requested by the users themselves, or they may be required by the IT infrastructure team to improve support of the system. During the postdelivery period, the IT infrastructure activities will include monitoring and evaluating the system, and proposing changes. Eventually, the system will be decommissioned, and in the lead-up to that point, data might be migrated to a new system. Activities postdelivery include:

- Use of the software;

- Development and installation of maintenance changes;
- Monitoring of the system infrastructure and service levels;
- Deinstallation and decommissioning.

12.1.1.3 Third-party developed system

The third party who developed the software will have a contract with the customer that describes the scope of their responsibility. It may be that they complete their responsibility when the acceptance test is complete, or when delivery is complete, or when some agreed-on period of time has passed (a warranty period), or they may have a long-term support contract. The activities might be the same as the COTS package, but perhaps in a larger organization the IT infrastructure group (supporters) would provide support for installation and a service/support line. It might be that the source code has been delivered and the supporters will carry out maintenance changes, or the third-party suppliers may make the maintenance changes. The activities would include:

- Use of the software;
- Development and installation of maintenance changes;
- Monitoring of the system infrastructure and service levels;
- Deinstallation and decommissioning.

12.1.1.4 System with periodic updates

Some software is delivered with an assumption of periodic updates; a good example is a virus checker. Here the vendor will supply upgrades at regular intervals; for example, monthly by mail or weekly via a download to the customer system. Typical activities include:

- Use of the software;
- Development and installation of upgrades;
- Deinstallation and decommissioning.

12.2 Delivery viewpoints

Once we have installed the software and it is in use, we will evaluate it. By "we," I mean any one who has an interest in the delivery. We ask: Does it help the customers? Are they happy with it? However, if only the views of the customers are considered in evaluating the software, we will find that we have not considered all the quality views (see Table 12.1).

In fact, a true evaluation of the software can only be made by involving all the groups, because although the project team has disbanded, all groups have an interest in quality at this stage. It is almost certain that the software will have maintenance changes, and these may involve work from the

Table 12.1 Views of Quality at Postdelivery—Simple

	Group
Quality View	*Customer*
Transcendent view	✓
User view	✓
Value view	
Product view	
Manufacturing view	

original development team as well as from maintainers working within the supporters group. Even with a COTS system installed by the customer, the builders, measurers, managers, and supporters at the vendor will have a view on the quality of the software.

For the evaluation, managers will be interested in measures of value for money and return on investment, supporting the value-based view of quality.

Supporters, measurers, and builders will be interested in measures of defects reported during use, supporting a manufacturing- and product-based view of quality. It is only once the software is live that we can evaluate the effectiveness of our defect prevention, identification and removal activities, including inspection, review, and testing activities. For example, test effectiveness could be measured by comparing the number of problems found in live use of the software with the number found during testing, and a similar measure could be made for reviews and inspections [1, 2]. If many more defects are found during QC processes such as testing and review than during live use, our quality measurement processes may be considered effective. We may additionally want to measure where we have made mistakes and introduced defects, in order to introduce improvements to reduce defects in future, as the fewer defects introduced, the better our planning and build processes were. A very simple example might help:

- Suppose we find 30 defects during our testing and reviews. In the first 6 months after delivery, we find another 20 defects. The total number of defects is 50. The efficiency of our defect finding processes is 30/50 = 60%.
- We analyze the 50 defects. Fifteen were introduced at the point where we gathered the requirements and were caused by misunderstandings between the customers and builders. We introduce some training and process changes to improve communication between the groups during requirement gathering, and set an expectation or target for a reduction in these types of defects in the next project. When we analyze the defects from that project, we will measure and compare to see if our process changes resulted in reduced defects.

Of course, that is a very simplified example. You would need to take onto account the other variables such as comparative size and complexity of the projects, and you might want to measure the cost rather than the number of defects. Of course, there may still be defects we have not found yet. Perhaps we would want to measure again after 12 months. All I want to indicate here is that it is possible to make measurements and use them to evaluate what work has been done as well as to drive improvement initiatives for the future.

Customers and supporters engage with the software day to day. They hold a transcendent view of quality ("are we enjoying this system?") but also the user-based view. They will have a perception of the software as fit for purpose or otherwise. They will evaluate whether the software has solved the problem they identified (Chapter 9) during start-up.

Involving all the groups gives a wider view of the quality of the system in use (Table 12.2).

12.3 Entry criteria for postdelivery

Postdelivery can only be entered if the delivery itself has been successful. If the delivery was rolled back or canceled, then the new software is not in operation. Example entry criteria are shown in Table 12.3.

12.4 Postdelivery—typical activities

12.4.1 Use of the system

This is the whole reason for the work up to the point of delivery. A software solution has been defined, designed, built, tested, and delivered. Finally, the

Table 12.2 Views of Quality at Postdelivery—Teamwork

	Groups				
Quality View	*Customer*	*Manager*	*Builder*	*Measurer*	*Supporter*
Transcendent view	✓	✓	✓	✓	✓
User view	✓			✓	✓
Value view		✓			
Product view			✓	✓	✓
Manufacturing view			✓	✓	

Table 12.3 Example Entry Criteria for Delivery

Example of Entry Criteria	*Example of Entry Documents*	*Example of Authorization Owners*
Installation and release complete, ready for live use	Delivery checklist complete and reviewed, updated IT service, updated configuration management system (new software, live version), known defect list, resolved defect list	Process owner, IT infrastructure manager

customers can use the software that will, we hope, resolve the problem or opportunity identified during start-up. The customers will use the software to carry out their tasks.

As we saw in Chapter 11, when software is first delivered, there may be problems, either with the delivery or with the delivered product. These will need to be resolved during the delivery period, either by fixing the problem or by working around it. Additionally, the users of the system will be new to it, and will need to learn how to use the system efficiently. This means that during the immediate postdelivery period, the customers may feel disillusioned. For example, I have recently taken delivery of a new laptop computer. My pleasure in unpacking the parcel with the shiny new machine had dissipated by two days later as I was still trying to install all the software, migrate my data, learn a slightly different keyboard layout, and get the laptop settings as I wanted. I was quite ready to throw it out of the window!

Once the bedding in of the software has been accomplished and the initial learning curve has been climbed, customers should start to gain benefits from the use of the new software. The organization as a whole will want to monitor that the cost of the software is outweighed by the benefit of using it. This should be measured over a period of time, taking into account the dip in enthusiasm (Figure 12.2 and [3, 4]).

Over time, the customers will require changes to the software, to correct problems, or to enhance the facilities available or to meet some change in requirements. They need to make a request for a change, and the change would be planned using the start-up activities we saw in Chapter 9. I will discuss changes to software in Section 12.4.3.

12.4.2 IT infrastructure and service management activities

The activities postdelivery include use of the software, monitoring of the system, evaluation of the system, maintenance and update of the system, and infrastructure support. The activities needed to support an organization's IT provision are covered in a number of best-practice guides, codes of practice, and standards, as we saw in Chapter 7; see, for example, those in [3–12]. If we talk to the supporters, we will find typically that they carry out

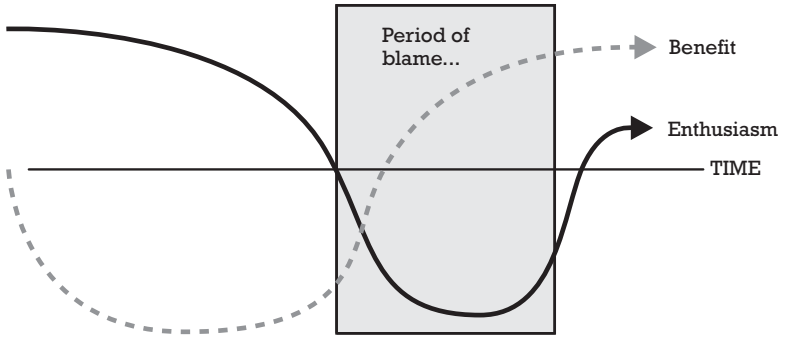


Figure 12.2 Silver bullet life cycle. (*From:* [3]. © 1998 DISC PD 0005:1998. Reprinted with permission.)

activities based on the recommendations of the IT Service Management Forum [7], which is an international center of expertise for IT service management. For example, the IT Infrastructure Library (ITIL), [4, 6, 8] and BSI Code of Practice for IT Service Management [3] identify activities for IT infrastructure management, service support, and delivery that focus on the customer's access to services that support the business functions, and the level and costs of those activities. These include:

- The *service, support, or help desk* is the customer's first port of call if something goes wrong; they rely on the other activities in this group.
- *Incident management* is intended to restore normal service as quickly as possible and minimize impact.
- *Problem management* seeks to minimize the impact of incidents and prevent recurrence of incidents.
- *Configuration management* identifies relationships between items that make up a system.
- *Change management* controls changes made to the configuration.
- *Release management* controls packages of changes as identified releases.
- *Service-level management* is about understanding, agreeing on, and monitoring the level of service provided by the IT system, perhaps by measuring against service-level agreements.
- *IT financial management* controls running costs for the IT systems, budgeting, accounting, and charging for services.
- *Capacity management* covers the monitoring and tuning of existing services, plus forecasting future requirements.
- *IT service-continuity management* is a subset of business continuity and defines a minimum set of IT provision for business continuity; for example, looking at whether during a change or release recovery/rollback is possible.
- *Availability management* looks at the factors from a customer viewpoint that are concerned with availability and reliability of the IT services.
- *Security management* defines and monitors the security of the IT systems and the software.
- *Infrastructure management*, including network service management, operations management, computer installation and acceptance, and systems management will continue throughout the life of the software and the IT system as a whole.

Note: If you are new to this subject area, or just need an overview, [9–11] are quick guides to the subject.

During the life of the software, changes to the overall system and the infrastructure will be needed. The supporters may need to upgrade hardware or software components that support the software the customers actually use. A start-up activity to analyze the problem would be needed, followed by

a project to provide a solution, which might be paper-based, software-based, hardware-based, or use some other media, or a mixture of these.

Many of the techniques we discussed in Chapter 9 for identifying and analyzing problems, for example, the Ishikawa fishbone, are specifically mentioned in ITIL [4] as tools to use postdelivery. Figure 12.3 shows an example of a partially completed fishbone diagram for customer complaints about the help desk; the complaints have been grouped under suitable headings, showing some of the underlying reasons (root causes) for the problems. In this example, the customers have complained that the help-desk staff are unaware of new software. During the session to discuss the complaints and put together the fishbone diagram, the help-desk team realized that they did not receive a known problem list for the new software. They did not know that the software had been delivered!

12.4.3 Making changes to an existing system

During the life of the software postdelivery, customers will identify new problems to be resolved and new opportunities to grasp, and will encounter new outside requirements. The system may need to be enhanced, corrected or adapted to changes in its environment. To accomplish this, supporters may need to have it changed to make future support and maintenance easier.

When changes are made to existing software, this is called software maintenance. It takes place in a software-maintenance life cycle (SMLC). Most software projects are SMLCs rather than SDLCs, because software spends most of its life in the postdelivery stage, being used and changed. Often, when people say they are working on a SDLC, they are actually working in software maintenance. So the system will be subject to successive cycles of change, often with many changes taking place in parallel (Figure 12.4). In fact, these are new SDLCs, involving the whole cast of characters from before, and could be described as SMLCs (software-maintenance life cycles).

There are four main types of maintenance changes to software systems:

- Corrective;
- Enhancement;
- Perfective;
- Adaptive.

12.4.3.1 Corrective maintenance

These are maintenance changes to rectify problems identified in products; making fixes to software, and correcting typographical or factual errors made in documents.

Corrections may be high-priority, even emergency, changes that need to be done at speed. Many corrections are not high priority or emergency, and can be built and released in groups or with other maintenance work.

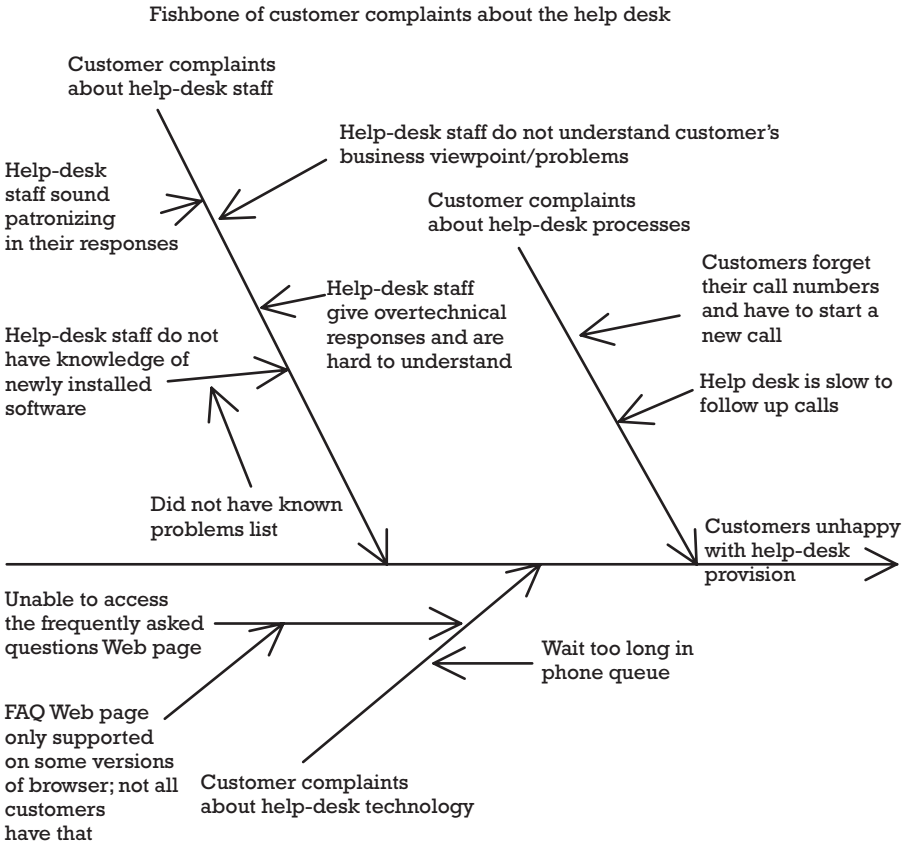
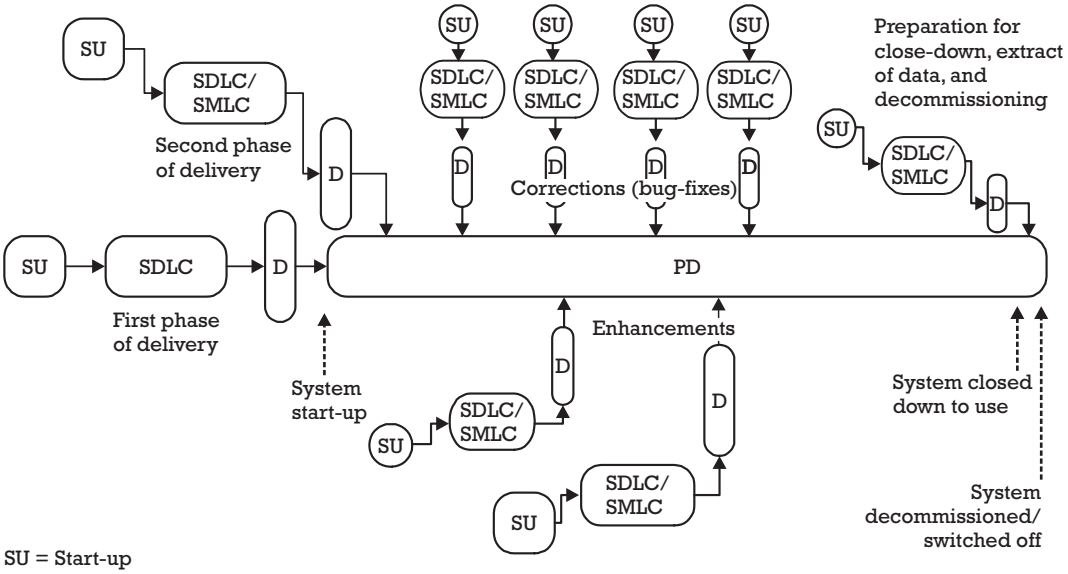


Figure 12.3 Example of an Ishikawa fishbone.



SU = Start-up
SDLC/SMLC = the software-development life cycle or the software-maintenance life cycle
D = Delivery
PD = Postdelivery

Figure 12.4 Parallel changes during postdelivery.

Even emergency changes that take place overnight need to be controlled within the life span structure I suggest. Suppose for example, that the overnight batch run stops unexpectedly, or that the company e-commerce Web site becomes unavailable. The maintenance to restore the service must take place as quickly as possible.

Despite the need for speed, the emergency change will require a start-up (problem and solution definition), some SDLC/SMLC activities, including requirements, design, build and test, and delivery. Because it is an emergency, a well-understood process with a suitable level of control and follow up is vital; it is during an emergency that we make mistakes. An SMLC designed as an "on one page" form with suitable checkpoints, tests, and sign-off may well be appropriate. It is important that the actions taken are documented. I am reminded of a notice I saw over a nurses' station in a hospital recently, "If it isn't documented, it didn't happen." Once the emergency is over, it is easy to forget what steps we took, and these might cause a chain reaction, resulting in other problems. Also, following the first-aid fix to the emergency problem, we may wish to revisit the changes, and perhaps even rework them to improve the fix.

Nonemergency fixes may be grouped and worked on in an agreed-upon timescale. Depending on the size of the fixes, it may be appropriate to use a tailored rather than a full SDLC, as we saw in Chapter 10.

12.4.3.2 Enhancements

Here the change is to enhance or improve some attribute of the software, either functional or nonfunctional. A group of minor enhancements would lead to a minor release and one or more major enhancements may lead to a major release, as we saw in Chapter 11. Enhancements and nonemergency fixes are often treated together; grouped according to priority, for example, and released in groups, in an agreed-upon timescale. Depending on the size of the changes, it may be appropriate to use a tailored rather than a full SDLC, as we saw in Chapter 10.

If we build and implement many small enhancements at one time, this might in itself be a major release in terms of risk; the size of the whole set of changes must be considered, as well as the impact of individual changes on "business as usual" and the possible interaction of changes.

One important point to consider is that there can be critical disagreement about whether changes are enhancements or correction of defects. This can be a contractual, even a legal issue. If a supplier organization has to absorb the cost of correcting defects within a fixed-price contract, it may resist accepting requirements problems as defects. The relationship between groups at this stage can deteriorate as arguments rage about whether a required change is a correction or an enhancement. If the customers are charged for enhancements but not for corrections, the argument from the builders and measurers may be that the software has been delivered to meet its specification, and if the customers require a change, this must be an enhancement. However, the customers may have interpreted

the specification differently, or the specification could be wrong. If we carry out the activities we discussed in Chapters 9 and 10, we should find that these differences of opinion are reduced. It is also important to have an agreement between all parties that defines what is considered a correction and what is considered an enhancement.

12.4.3.3 Perfective maintenance

This is work done to make the software more maintainable in future. This is important to reduce support and maintenance costs. Typical problems with software that is changed frequently and used over a long period of time include:

- The people who know the software and who know the reasons why it has a certain structure leave, taking knowledge with them.
- The documentation that describes the system (requirements documents, design specifications) is not updated with the software, so it is out of date and does not describe the software.
- The code becomes increasingly complicated as it is changed over time, and, eventually, it is not possible to tell what is affected by a change.
- The complexity and lack of definitive information make it increasingly difficult to test the changes to the software, especially as software tends to become more complex as it is changed over time.

In one organization, I remember that a package had been customized for particular clients, but each client's software was built on some central code. One of the programmers, in making a change to a client area of the code, changed code that affected all the clients, without realizing it. It was so difficult to understand the code that this mistake was not surprising; the software had grown almost organically over the years, and was not structured for easy maintenance. It was also impossible to know that the testing had covered the changes and the rest of the software sufficiently. The software needed to be completely restructured to make it possible to maintain and test it.

The activities in perfective maintenance include rewriting code to improve its structure (reengineering the code) and writing specification and design documents that describe the existing system (retrodocumentation).

12.4.3.4 Adaptive maintenance

These maintenance changes are made to support changes in the software's environment; for example, it may be necessary to upgrade hardware or systems software, or other supporting parts of the IT infrastructure. The software may need to be changed—adapted—to work properly in its new environment. This may include "porting" software or migrating data to a new infrastructure, in which case the software or the data or both may need to be adapted. An example would be my new laptop; moving my databases onto

the new laptop was easy—copy to and from removable media. However, in order to use the data in the database, I had to make a choice either to allow the upgraded database software to reformat the data, in which case I would not be able to use it on the older laptop, or to use it as read-only. The new software was going to carry out adaptive maintenance on the data to change its format.

12.4.3.5 Who gets involved in a change?

All the groups have an interest in the changes made:

- *Customer:* How will this change affect me? What is its impact? What are the implications for my normal work? Will it affect business as usual? Will it benefit me? What are the risks? Do I want this change?
- *Supporter:* How will this change affect me? What is its impact? What are the implications for the IT infrastructure and for service management? Will it benefit me? What are the risks? Do I want this change?
- *Manager:* What are the cost implications of making the change and of not making the change? Will there be a cost–benefit trade-off? If we do this work, what other work has to be halted? What is the priority of this change compared with others?
- *Builder:* Will I be asked to make the change? Do I want to make the change? Do I understand the change?
- *Measurer:* Will I be asked to test the change? What else needs testing? Can I test the change?

Whatever the size of the change we want to make, we need to involve all the groups and go through the activities described in Chapters 9, 10, and 11. For a small change we would do a "cut-down" version of the activities—the checklist approach we have looked at in the earlier chapters. For a large change we would take a more detailed approach. Even an emergency fix requires involvement from the different groups, to get an understanding of the impact of the problem and the proposed fix. In one organization, the overnight call went not just to the analyst/programmer who would implement the change, but also to a senior business user who would assess the impact on business as usual of the proposed fix, compared with the impact of the problem itself.

12.4.3.6 Testing changes to software

Testing changes to existing systems is difficult. Because it is easy to make changes that have an unexpected effect on software, we want to make sure that the change works, and that other parts of the software have not been changed. The assessment of the impact of any proposed changes must be made both in terms of what products are effected—"If this screen is changed, the user documentation and the training material also need changing"—and also in terms of what measurement and assessment is

needed—"If this screen is changed, I need to test that the change is correct in all those products, and also to check that these other related parts of the software and other products have not changed." This is known as *regression testing*; we need to check that we have not changed anything we should not have changed. The need for regression testing means that changes that look simple may be expensive; the requirements, design and build part of the SMLC may be easy and cheap to do, but the testing, including a regression test, may take a long time and be difficult to plan and execute. I do not want to get into the detail of regression testing here, but it must be considered when considering the impact of a change:

> You are right to emphasize the testing task. I remember one major bank saying that whenever they had made significant changes to their core banking system, they always had to run a regression test that cost several million dollars, and took several weeks to complete. (Richard Warden, personal communication, July 2003)

There is no reason to suppose the burden will decrease with Web site and e-commerce work; continuous change will mean continuous regression testing of very public functional and nonfunctional attributes.

12.4.4 Monitoring and evaluation

During the postdelivery period, we can monitor the use of the software and collect information that will help us measure the success of the original SDLC and of subsequent SMLCs. We can also monitor and evaluate the service management and infrastructure management services.

Because the postdelivery period of the software is the longest part of its life span, we will want to look for improvements in the software itself and in the surrounding support services. ITIL [4], for example, suggests use of the *Deming Cycle*, proposed by W. Edwards Deming [13]. The cycle has four stages, sometimes called the Plan, Do, Check, and Act cycle and sometimes the Plan, Do, Review, and Improve cycle (see Figure 12.5). Here, we plan what to do, and we do it. Then we review what we did. Was it successful? Did it go as planned? What should we improve? We then put improvements in place and plan the next cycle of activities.

12.4.4.1 Evaluation of the SDLC/SMLC

A formal evaluation of each SDLC/SMLC is important; this should involve representatives of all the groups. The evaluation needs to be timed sensitively, and may need to be a series of evaluations over a period of time. I remember one project manager saying, "Never have the end-of-project party until six months after the software's been installed!" Table 12.4 shows some possible timings for evaluation and the advantages and disadvantages of each.

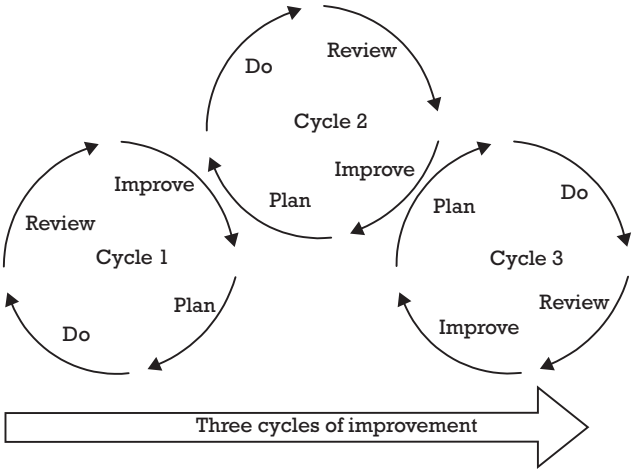


Figure 12.5 Plan, Do, Review, and Improve cycle. (*After:* [13].)

Evaluation meetings need to be managed carefully, so take into consideration of some of the team and communication techniques we have discussed in earlier chapters. Use the Six Hats [14] technique described in Appendix A, for example, to help participants look for good points as well as bad points for the evaluation (Table 12.5).

12.4.4.2 Ongoing monitoring and evaluation of software

Part of the service delivery work will include monitoring the software and surrounding services, including numbers of problems reporting to the help desk or support lines. These should be used to analyze whether:

- SDLC/SMLC software/code products could be improved, for example, if problems are being reported that could be resolved by better design, or might have been found with improved testing.
- Other SDLC/SMLC products could be improved, for example, if a task is described in the on-line help or user guide, but customers cannot find

Table 12.4 Timing the Postdelivery Evaluation of the SDLC/SMLC

Time After Delivery	*Advantage*	*Disadvantage*
Immediately	Will not have forgotten the good and bad points of the SDLC/SMLC.	Too soon to tell if the software works in reality.
During the immediate postdelivery period	Will not have forgotten the good and bad points of the SDLC/SMLC.	May be too busy. May be too soon to assess.
During low point of period of blame (Figure 12.2)	Will not lose information about the problems and areas for improvement.	Will be only focused on bad points. Evaluation meeting likely to be unpleasant.
Six months after delivery	Can do cost–benefit analysis against a period of use. Successes and problems seen against perspective of real use.	Too late to influence the following projects. Team now focused on other projects. May have forgotten both problems and what was done well.

Table 12.5 Techniques Used to Aid Teamwork Postdelivery

Subject Area	*Technique Examples*	*In 30 Seconds… (see Appendix A for more)*
Team relationships and natural roles/team skills	Belbin Team Scores [15]	Support of the software can fail because of personal rather than technical factors. Teams need to understand their strengths and weaknesses as a team. A balance of roles/skills is required in the personalities in the team. Example roles: plants have new ideas; completer–finishers want to finish to fine detail. Too many plants and you will never finish anything.
Improve communication—empathy with others	MBTI [16] Honey & Mumford [17] Kirton [18]	Different people have different personalities and communication styles. People who wave their arms around and talk a lot can annoy people who like to be quiet and think, and vice versa. The Myers-Briggs Type Indicator (MBTI) identifies four contrasting type pairs, e.g., *Introvert/Extrovert*, leading to 16 "types" (e.g., INTJ is *Introvert-iNtuitive-Thinking-Judging*). The Honey & Mumford Learning Styles Questionnaire identifies preferred learning styles (e.g., *Pragmatists vs. Theorists*) require different experiences to learn. Kirton identifies preferred problem-solving methods (*Adaptors vs. Innovators*)—do we break the rules or work within them?
Improve meetings	De Bono's Six Thinking Hats [14]	Improve meetings by setting rules for behavior. Six "hats" are used. Everyone wears the same color hat at the same time: Blue Hat—meeting structure, Black Hat—pessimistic, Yellow Hat—optimistic, Red Hat—feelings, White Hat—facts, Green Hat—creative ideas. Allows meeting members to move outside their stereotypes and allows time for different, sometimes difficult types of communication.
Identify problems and root causes, find solutions	Ishikawa fishbones [4, 19]	Use to identify problems, root causes of problems and solutions. On a fishbone diagram, brainstorm problems, their possible causes, their root causes, and, therefore, solutions to the root cause.
Review processes	Reviews [2, 20]	For processes use audit. For checking process documents use walkthrough, peer review, or inspection.
Understand whether an idea is worth pursuing	Risk workshop by brainstorming [19, 21]	A brainstorming workshop is run to list all the possible risks people can identify. The risks are sorted into groups, separating risks (might happen in the future) from issues (problems right now) and constraints. The risks are scored for impact and likelihood, and ranked to give a prioritized list.
	Cost–benefit analysis [4, 21]	Cost–benefit analysis is done by calculating the predicted benefits of the proposed change (time, money saved) and setting this against the predicted cost. An example of a worked cost–benefit analysis for service support is in [4], a general worked example is in [21], and there is also an example in Appendix A.

the information, it does not mean they are foolish; it means the user guide and on-line help should be improved.

- SDLC/SMLC software/code and other products are not deteriorating over time; for example, as changes are made to code, it can become more complex, until it is far too expensive to change and test successfully. Tools are available for measuring software complexity, so this can

be monitored over time. Additionally, other products can become out of step with the code, and this can be monitored through configuration management if it is applied to all products and not just the code.

- Service support and delivery could be improved. For example, if customers find they are waiting too long for a response from the help desk, what could be done to improve call turnaround?

12.4.4.3 Monitoring and evaluating the postdelivery processes

This can be done by carrying out process audits [20] to check that the defined processes are being followed and that they are suitable. ITIL suggests use of the EFQM Excellence Model or its U.S. equivalent, the Baldrige criteria [4, 22, 23]. As we saw in Chapter 1, EFQM would expect evaluation of measured performance and perceptions of the postdelivery services from the viewpoint of the *Customers*. *Society* results measure what the organization is achieving in relation to local, national and international society, where appropriate—the wider customer group. It would also expect measurement of *People* (those who are employed to carry out the work, in this case, the builders, measurers, supporters, and managers). EFQM expects measurement of the *Key Performance Results*, including financial measures such as return on investment, which will interest the managers and customers.

12.5 Exit from postdelivery

We only exit from postdelivery when the software is deinstalled (removed) or the system closes down. This may never happen; some software is more than 20 years old, and if it is still working, there is no real reason to remove it. Some systems and software are deinstalled or decommissioned. The exit from postdelivery will generally take place when the replacement software is available. Typically, a full SDLC (Chapter 10) and delivery will have completed successfully before the old software is removed and the old infrastructure decommissioned. It may be that the old and new software are run in parallel for a while; the same tasks being carried out on both until the customers and supporters are satisfied that the new software and system has been delivered correctly. In Table 12.6, we see examples of exit criteria for postdelivery.

12.6 Conclusion

In Chapters 8 to 12, we have followed a software system through its life span, from start-up, through the SDLC, into delivery, and so into the activities of postdelivery, including repeated SMLCs. In summary the activities are:

- *Start-up:* We explore the problem until we understand it, decide whether it is worth solving, set general constraints and parameters for

Table 12.6 Example of Exit Criteria from Postdelivery

Example of Exit Criteria	*Example of Deliverables Documented, Reviewed, and Agreed Upon*	*Example of Authorization Owners*
System ready to decommission: new system accepted, data migrated	Acceptance criteria sign-off list. Accepted new system (SDLC exit and delivery exit). Data migration controls sign-off. Parallel run sign-off.	Business process owner, infrastructure manager
System decommission complete	Data migration complete. Parallel run complete. Software and data back up and archive complete. Software close-down checklist complete. Hardware close-down checklist complete.	Business process owner, infrastructure manager

the solution, decide how to approach the problem, for example, a software solution may be appropriate, and agree on a formal or informal contract of work, with constraints and acceptance criteria and an outline or high-level plan.

- *SDLC:* We plan the SDLC in detail and monitor the activities, we manage change, we define the requirements in detail, we design products, including the software code, we build the software based on the designs, and we test to see that the software and other deliverables are OK.
- *Delivery:* We install the software and check that it has installed correctly.
- *Postdelivery:* We use the software, carry out service delivery and support activities, carry out IT infrastructure activities, make changes to the software, and monitor and evaluate the postdelivery activities to look for improvements.

At each stage, all the groups must contribute; at each stage customers, managers, builders, measurers, and supporters help enable software quality to be built into the software and to be confirmed. We saw in Chapters 3 to 7 that each group holds its own view of quality, and that all these views are reasonable and must be considered.

No single standard will help right through the software life span. The EFQM Excellence Model [22] will provide a framework for all activities, and with techniques like Balanced Scorecard [24], enable a strong alignment between the goals of IT and its customers, perhaps in one integrated management system. Within that elements of CMM® [25] and its relations, such as the Team and Personal Software Processes [26, 27], ISO 9000 [28] will enable us to control our work during the SDLC and help us decide which SDLC model we use, whereas the ITIL [4] will provide a structure during the deployment and optimization of the software as it is being used and changed.

Teamwork and communication is fostered by clear processes, which encourage involvement from all the groups, and recognize their different viewpoints. Clear entry and exit points agreed on by all the groups will help this. I mentioned W. Edwards Deming earlier; he distilled his philosophy of quality management into 14 points [13]. In the ITIL [4], a number of these points are highlighted, and at the top of their list is a key point: "Break down barriers—improve communication between departments" [4, 13].

Let me take this further: Break down barriers between people; communication and team work are key to achieving software quality. It is not my responsibility, and it is not your responsibility. It is our responsibility, together.

References

[1] Fewster, M., and D. Graham, *Software Test Automation*, Reading, MA: Addison-Wesley, 1999, pp. 211–219.

[2] Gilb, T., and D. Graham, *Software Inspection,* Reading, MA: Addison-Wesley, 1993, pp. 386–388.

[3] British Standards Institute, DISC PD 0005:1998, *Code of Practice for IT Service Management,* 1998 (PD0005).

[4] IT Infrastructure Library, *Best Practice for Service Support,* London, England: Office of Government Commerce, 2002.

[5] IT Service Management Forum, "BS 15000 IT Service Management," http://www.bs15000certification.com, October 2003.

[6] IT Infrastructure Library, *Best Practice for Service Delivery,* London, England: Office of Government Commerce, 2002.

[7] IT Service Management Forum, "IT Service Management Forum," http://www.itsmf.com, October 2003.

[8] IT Infrastructure Library, "IT Infrastructure Library"; see the ITIL Web site http://www.itil.co.uk or the TSO Web site http://www.tsonline.co.uk/bookshop/bookstore.asp?FO=1150345, October 2003.

[9] IT Service Management Forum, *A Dictionary of IT Service Management: Terms, Acronyms and Abbreviations: Version 1,* London, England: The Stationery Office, 2001.

[10] IT Service Management Forum, *IT Service Management: A Companion to the IT Infrastructure Library: Version 2,* London, England: The Stationery Office, 2001.

[11] IT Service Management Forum, *A Dictionary of IT Service Management: Terms, Acronyms and Abbreviations: Version 1 (North America),* London, England: The Stationery Office, 2001.

[12] British Computer Society, BCS Qualifications, http://www1.bcs.org.uk/link.asp?sectionID=574, September 2003.

[13] The W. Edwards Deming Institute, "Deming's Teachings," http://www.deming.org/theman/ articles/articles_gbnf04.html, November 2003.

[14] de Bono, E., *Six Thinking Hats®*, New York: Penguin, 1999.

[15] Belbin Associates, "Belbin Team Roles," http://www.belbin.com/belbin-team-roles.htm, October 2003.

[16] Team Technology Web site, "Working Out Your Myers Briggs Type," http://www.teamtechnology.co.uk/tt/t-articl/mb-simpl.htm October 2003.

[17] Honey, P., "Learning Styles," http://www.peterhoney.co.uk/product/learning styles, October 2003. PeterHoney.com, 10 Linden Avenue, Maidenhead, Berks, SL6 6HB. Tel.: 01628633946. Fax: 01628633262. E-mail: info@peterhoney.com.

[18] Kirton, M. J., "Adaptors and Innovators Defined," KAI Web site, http://www.kaicentre.com/, July 2003.

[19] TQMI, *Problem Solving—Tools and Techniques*, Frodsham, England: TQMI, 2001.

[20] IEEE 1028™ Standard for Software Reviews, 1997.

[21] Robson, M., *Problem Solving in Groups*, Aldershot, England: Gower, 1995.

[22] European Foundation for Quality Management, "EFQM Excellence Model," http://www. efqm.org, August 2003.

[23] Malcolm Baldrige model, http://www.quality.nist.gov/index.html, August 2003.

[24] Kaplan, R. S., and D. P. Norton, *The Balanced Scorecard*, Boston, MA: Harvard Business School Press, 1996.

[25] Software Engineering Institute, "Capability Maturity Model®," http://www.sei.cmu.edu, July 2003.

[26] Humphrey, W., *Introduction to the Team Software Process,* Reading, MA: SEI, 2000.

[27] Humphrey, W., *Introduction to the Personal Software Process*, Reading, MA: SEI, 1997.

[28] International Standard Institute, ISO 9000:1994 and ISO 9000:2000 Quality Systems.

Selected bibliography

Belbin, R. M., *Management Teams—Why They Succeed or Fail*, London, England: Butterworth Heinemann, 1981.

Deming, W. E., *Out of the Crisis*, Cambridge, MA: MIT Press, 2000.

Honey, P., and Mumford, A., *The Learning Styles Helper's Guide,* Maidenhead, England: Peter Honey Publications, 2002. PeterHoney.com, 10 Linden Avenue, Maidenhead, Berks, SL6 6HB. Tel: 01628633946. Fax: 01628633262. E-mail: info@peterhoney.com.

IT Infrastructure Library, *Best Practice for Application Management,* London, England: Office of Government Commerce, 2002.

IT Infrastructure Library, *Best Practice for Security Management,* London, England: Office of Government Commerce, 2002.

Kroeger, O., J. M. Thuesen, and H. Rutledge, *Type Talk at Work: How the 16 Personality Types Determine Your Success on the Job,* New York: Bantam Doubleday Dell, 2002.

McHale, J., "Innovators Rule OK—Or Do They?" *Training & Development,* October 1986, http://www.kaicentre.com/.

Quagliariello, P., "Introduction to IT Service Management, ITIL and ITIL Capacity Management," http://www.pultorak.com/home/speaking_engagements/presentations/ 2003_03_07_cmg.pdf, October 2003.

Appendix A

Techniques and Methods

This appendix is summary of the techniques mentioned in the chapters, with additional information and resources for you to follow up ideas that interest you. If you start a literature or Internet search for any of these topics you will find much more information and other related or similar techniques. This appendix is intended to introduce you to ideas and resources rather than to be a complete compendium of the techniques available.

A.1 Communication, team dynamics, and meeting behavior

We all carry with us *taken-for-granted* assumptions about the world and other people—what is normal and how people should behave—and it these assumptions that allow us to build trust between individuals within groups and organizations [1]. Although many taken-for-granted assumptions are shared, there will be differences between cultures, organizations, and family/friendship groups. Just because you "always do it that way" it does not mean it is the right way. One of our assumptions can be that others will share our taken-for-granted assumptions. If we discover that other people have different core beliefs, we are surprised and may dismiss their views as wrong—it is hard to see other people's view. We may also assume that other people communicate in the same way as we do, that they share our sense of humor, use body language in the same way, and share our preference for written, pictorial, or verbal messages. I recently had a discussion with a colleague because we could not understand each other. It turns out that we used two key phrases in opposite ways, both equally correct but meaning the opposite of each other:

- I had used "I think" to mean "I am not sure" but my colleague had used "I think" to mean "I am certain."
- My colleague had used "I feel" to mean "I am not sure" but I had used "I feel" to mean "I am certain."

These communication differences are increased by cultural differences between organizations, and between colleagues from different types of organization, and from different countries. We need to be careful of our own taken-for-granted assumption, and be aware of other people's without dismissing them. This section is an introduction to some of the techniques and ideas people have developed to help understand and bridge communication gaps.

In this appendix, I will introduce a number of ways of examining communication preferences. It is very important that to get the most from these techniques, you use this as just a taster. I have suggested search words to help you find out more from the Internet, and there are books, journals, and Web sites in the references and selected bibliography. Be aware also that you will not get the most from some of these techniques without the help of specialists; psychometric tests are not to be played with.

For this section, the Internet/Web search words are: taken-for-granted assumptions, communication skills, team work, ethnomethodology, sociology, team dynamics, body language, transactional analysis, personality types, culture, Garfinkel, Busco, emotional intelligence, and Goleman.

A.1.1 Belbin Team Roles

Whether teams are drawn from all the groups (customer, manager, builder, measurer, supporter) or from a single group, work can fail to be successful if people in the group do not understand their strengths and weaknesses as team members. This means we may fail to deliver and support software successfully because of personal rather than technical factors. Teams need to understand their strengths and weaknesses *as a team.* Belbin [2–4] defined a number of roles that people take when in a team. These are summarized in Table A.1.

A balance of roles/skills is required in the personalities in the team if it is to be successful. For example, *plants* have new ideas whereas *completer–finishers* want to finish to fine detail. If you have too many *plants* in the team you will never finish anything, but if you only had *completer–finishers*, you might not have so many new ideas. Each person in the team takes an assessment, and this provides a picture of the strengths and weaknesses of the team as a whole. All the roles are needed, so if one is missing, either the team fails or some team members take on secondary roles. I have certainly observed that people will change roles depending on the particular team, but they may not be comfortable. Most people will score to show a mix of the team roles.

Assessment is by a questionnaire that is administered and marked by a Belbin assessor. Details of how to get a Belbin assessment for a team can be found on their Web site [4]. Assessments may be on-line or by mail. They also provide training for certification in use of Belbin assessments. It is best to use qualified people to administer these types of tests.

The earlier work by Belbin [2] has been updated, and the latest thinking is in [3, 4].

Table A.1 Belbin Team Roles

Team Role	*Contribution*	*Allowable Weakness*
Plant	Creative, imaginative, unorthodox	Ignores incidentals
	Solves difficult problems	Too preoccupied to communicate effectively
Resource investigator	Extrovert, enthusiastic, communicative	Overoptimistic
	Explores opportunities	Loses interest once initial enthusiasm has passed
	Develops contacts	
Coordinator	Mature, confident, a good chairperson	Can be seen as manipulative
	Clarifies goals, promotes decision making, delegates well	Offloads personal work
Shaper	Challenging, dynamic, thrives on pressure	Prone to provocation
	The drive and courage to overcome obstacles	Offends people's feelings
Monitor Evaluator	Sober, strategic and discerning	Lacks drive and ability to inspire others
	Sees all options	
	Judges accurately	
Teamworker	Cooperative, mild, perceptive and diplomatic	Indecisive in crunch situations
	Listens, builds, averts friction	
Implementer	Disciplined, reliable, conservative and efficient	Somewhat inflexible
	Turns ideas into practical actions	Slow to respond to new possibilities
Completer–finisher	Painstaking, conscientious, anxious	Inclined to worry unduly
	Searches out errors and omissions	Reluctant to delegate
	Delivers on time	
Specialist	Single minded, self-starting, dedicated	Contributes on only a narrow front
	Provides knowledge and skills in rare supply	Dwells on technicalities

In this section the Internet/Web search words are: Belbin, team, team dynamics, and team roles.

A.1.2 De Bono's Six Thinking Hats

Sometimes in meetings, we either find that the same people always contribute the in the same way—"Joe's always so pessimistic and negative" or "Mary never acknowledges there might be problems"—and sometimes the meeting dissolves into acrimonious chaos, with people no longer on speaking terms. Can we do anything about this? Edward de Bono [5] decided that we could, and the Six Thinking Hats was the result. I first came across it as an idea for improving software projects from Jens Pas' EuroSTAR presentation and workshop [6] as a means of introducing emotional intelligence to rationally biased software projects; in other words, allowing fact-based people an opportunity to also express emotional views.

Use of de Bono's Six Hats at a meeting encourages people to take different roles at the meeting, and to categorize their communication and thus improve it. This means that it is acceptable to express emotion, for example, but we separate those communications from the other communications, such as collection of facts. The six hat colors and their meaning are summarized in Table A.2.

By using the hats, we set rules for behavior. Everyone wears the same color hat at the same time, and the hats are not associated with particular

Table A.2 Six Hats at a Meeting

Hat Color	*When we ask everyone to wear this color hat, it means the meeting will focus on*
White	Providing facts—what information do we have? Do we know it is fact, or is it opinion?
Red	Adding to the White Hat thinking by expressing our feeling about the situation we are discussing and working from intuition, hunches, opinion—what do we feel? What is our "gut reaction"? Can we identify sensitivities?
Black	Identifying what could go wrong—should we be cautious about this? What risks are there? What have we overlooked? What if?
Yellow	Looking for the positive things—what benefits are there? What new proposals do we have? How can we make this happen? What if? Leading to …
Green	Creating new ideas—supposing we tried…? We could do it this way instead!
Blue	Thinking about the process of the meeting itself—was that what we meet to cover? Which hat would be useful now? Have we spent long enough in *Black Hat*?

people. This allows meeting members to move outside their perceived stereotypes and allows time for different, sometimes difficult types of communication. De Bono discussed the order and way the hats are used; it depends on the meeting and the problem, as well as the team's experience in using the techniques. Suppose we have a meeting to discuss a design for an interface. We might have a meeting agenda like this:

- Blue Hat to set the scene—why are we here and what do we want to achieve?
- White Hat—what facts do we have about the design that every one can agree on?
- Red Hat—do we like or dislike the design—what is our gut feel about it?
- Yellow Hat—what are the good things about the design?
- Black Hat—what disadvantages can we see in the design?
- Green Hat—can we identify new ideas to help us overcome the black hat points?
- Red Hat—how do we feel now about this design?
- Blue Hat to close the meeting—what are the next steps?

The advantages that de Bono identifies [5] are:

Powerful, focused working which is

- *Time saving* because the meeting is not run as a series of confrontational arguments, thus
- *Removing of ego* by removing confrontation allowing the meeting to deal with
- *One thing at a time*.

All the ideas from all the people at the meeting at treated as parallel rather than confrontational—once a complete picture has been arrived at using the Six Hats then it is easier for people to agree a solution or make a decision. We can use the hats to change the focus of the meeting: "It looks like with the Yellow Hat thinking we have identified some real advantages to the approach we're discussing. Let's just try some Black Hat thinking—what are the disadvantages?"

You can find more information about de Bono's work on thinking, including the Six Thinking Hats, on his Web site [7].

In this section the Internet/Web search words are: de Bono, Six Hats, Meetings, and team work.

A.2 Communication styles

In order to improve communication with others, we need to understand their points of view and communication styles. We need to empathize with others. I do not intend to pigeonhole people by discussing these indicators of communication styles, but I think it is useful to consider that our own preferences are not "right"—they are just preferences. My observation of people is that their preferences will change with mood, context, and group mix.

A.2.1 Myers-Briggs Type Indicators

The work of Myers and Briggs, described, for example, in [8–11], suggested that different people have different personalities and communication styles. To put it at its simplest, people who wave their arms around and talk a lot can annoy people who like to be quiet and think, and vice versa. Myers and Briggs built the *Myers-Briggs Type Indicator* (MBTI), which identifies four contrasting type pairs:

- *E and I: E*xtroversion (e.g., prefers social interaction) versus *I*ntroversion (e.g., quiet and private)
- *S and N: S*ensing (e.g., experience and practicality) versus i*N*tuition (e.g., prefers novelty and aspiration)
- *T and F: T*hinking (e.g., prefers analytic or critical approach) versus *F*eeling (e.g., prefers sympathizing and appreciative approach)
- *J and P: J*udgment (e.g., prefers firmness and control) versus *P*erception (e.g., prefers flexibility and spontaneity).

There are other factors which make up the pairs, as you see if you look at [8–11]. These four pairings give rise to 16 "types" (e.g., INTJ is Introvert-iNtuitive-Thinking-Judging), as summarized in Table A.3.

We discussed briefly in Chapter 5, how these communication preferences might affect the communication between builders and other groups. Conflicts arise between any two people, regardless of group. You can see

Table A.3 Myers-Briggs Type Indicators: The 16 Types

ESTJ	ESTP	ESFJ	ESFP
ENTJ	ENTP	ENFJ	ENFP
ISTJ	ISTP	ISFJ	ISFP
INTJ	INTP	INFJ	INFP

that if two people have preferences at different ends of the table, there is real potential for them annoying each other.

Research [10] has shown that more IT people tend to be INTJ than in the general population, where a larger proportion tend to be ESTJ or ESFJ:

> INTJs represent just 2% of the population, yet form about 10% of computer staff. The other main types in computing are ENTJ, ISTJ, and ESTJ. (The latter two feature highly in many jobs, as they represent a large part of the male population). ESTJ is a likely culture of the business units you are supporting. [10]

The communication preferences of these groups may be radically different.

If you are going to look at MBTI, beware of stereotyping! It is worth noting that just because someone is in a particular job, it does not mean they are a particular type, and just because someone appears to be a particular type does not mean they will be a success in an apparently related job. Remember, if 10% of computer staff are INTJ, then 90% are *not* INTJ. Similarly, being a woman does not mean you *cannot* be an ESTJ.

As with all of these types of test, make sure it is administered by someone who is suitably trained and can interpret the results properly.

In this section the Internet/Web search words are: Myers-Briggs Type Indicator and MBTI. Also try searches on particular types such as INTJ and ESFJ, since there are Web sites and discussion groups for some of the types.

A.2.2 Honey and Mumford Learning Styles

When we want to impart new information, we design user guides, help messages, wizards, training sessions, and presentations to help us put our message across. We need to think about the communication styles of our audience, and as part of that we need to understand that *different people learn in different ways* and there is not one *right way* to teach or to learn. Honey and Mumford's work identified people's preferred learning styles [12, 13]. There are four styles identified by Honey and Mumford, and you will find the learning styles concepts adopted by education and training organizations, for example, [14]. The learning styles identified are:

- Reflectors like to collect data and think about it, reflect, and observe, but are not so happy working spontaneously without time to prepare.

- Activists like to do things, work in teams, and try things out, but they learn less from teaching by lectures or reading on their own.
- Theorists like new ideas and enjoy adapting and integrating observations into complex and logically sound theories, but they do not enjoy situations that emphasize emotion and feelings, or unstructured activities with poor briefing.
- Pragmatists like training that is related to their immediate job or problem, where they have a model to follow, but they do not like training with no apparent payback to the learning, or if they see the event or learning is "all theory."

This means that when we need to get new ideas across to people, we need to think about two things:

- Who needs to know about what and can we identify their preferred learning style? For example, not everyone will learn from a slide presentation.
- Who would be good at delivering the information in that style? For example, not everyone is comfortable with teaching a role play/group work session.

If we are implementing a large, new system affecting many people we may need to offer several different ways for people to learn about the system itself, when and how it will be implemented, and how to get help and further information. We may even want to consider how we organize QA and QC measurement activities using these ideas; activists and pragmatists may find that a review of a working prototype, perhaps with a role play rehearsal, helps them to identify potential problems, and perhaps theorists and reflectors may get more from an inspection process.

The importance of learning and creativity in management is being increasingly recognized and other theories have been advanced, but as there is evidence that the Honey and Mumford approach links with the MBTI we discussed in Section A.2.1, and it is designed specifically to link to organizations and management [15], I have just described this one approach.

The Learning Styles Helper's Guide and *The Learning Styles Questionnaire 80 Item Version* (containing the questionnaire) are copyrighted by and available from Peter Honey Publications. The questionnaire is also available on-line at http://www.peterhoney.com or from Peter Honey Publications Limited, 10 Linden Avenue, Maidenhead, Berks SL6 6HB. A description of the learning styles is reproduced by permission of Peter Honey [13].

In this section the Internet/Web search words are: Honey, Mumford, learning styles, reflector, activist, theorist, pragmatist, and training styles.

A.2.3 Kirton adaptors and innovators

Kirton's work on *adaptors* and *innovators* looks at two groups of characteristics that each of us have to a greater or lesser degree; we are a mix. Adaptor

characteristics are those of discipline, efficiency, and resolving problems within the existing rule set and with an eye to maintaining the team's cohesion. Innovators challenge the rules, and may appear to adaptors to be insensitive to needs of the team; they enjoy radical change and challenging the accepted rules. This theory is not about ability; it assumes all people are creative and instead asks "Will someone prefer to be creative within the rules or by breaking the rules?" We all find ourselves somewhere on the continuum between extreme adaptor and extreme innovator:

> The beauty about the KAI is that it makes no negative evaluations about people. Whatever point you find yourself on along the continuum, you will find advantages and disadvantages associated with being there. Indeed, what will be useful and appropriate (advantageous) in one situation might cause problems in another. Armed with this insight, it then becomes possible for people to make informed choices about how they cope with the situations they find themselves in. [16]

The KAI characteristics are measured by a short test; information about this is on the KAI Web site [17]. Again, it needs to be properly administered and interpreted by a trained and certified KAI administrator.

In this section the Internet/Web search words are: KAI, Kirton, adaptor, innovator, and creativity.

A.2.4 Motivation studies

"We have sent them on a training course, we have bought them new PCs and we paid them a bonus—and they are still miserable! What's wrong with these people?" That director of a software company had motivational problems in his teams, and just could not solve them. What was wrong?

I have mentioned in a number of chapters the work on motivation done by Warden and Nicholson [18], which is based on the Job Characteristics Model of Motivation [19]. The job diagnostic survey provides a comprehensive set of motivational measures. As a process model, it can diagnose problems with motivational dynamics caused by poor job design. Psychometric measurement techniques do not provide this capability.

A number of other studies into what motivates people have been carried out; see [11, 20] for some examples. One commonly used is based on work by Maslow in the mid-twentieth century [21], which developed into the *Maslow Hierarchy of Needs,* described in many books and Web sites, for example, references [22–24]. Maslow's original levels of need may be described as:

- *Physiological*—food, water, sleep;
- *Safety*—freedom from danger;
- *Love*—need to belong to a group, friendship;
- *Esteem*—reputation beyond family and friends;
- *Self-actualization*—self-fulfillment.

The model has developed over the years, by different authors, so if you look at this, particularly by Internet searches, you may see the hierarchy extended or detailed added to explicitly mention cognitive, aesthetic, and spiritual needs as well as the need to help others.

Until someone's basic needs (for example, food and shelter) are met, this may be all they are interested in. Once they have enough at one level, then other motivators become more important —however much more food you give them, they will not be any happier. The cutoff point for moving from one point in the hierarchy to another is different for different people, so for some people "enough" at one level is different from another person's need at that level.

When we look at these two studies together with the personality traits we have examined above, we can see that there may be many reasons why our director's teams might still be unhappy and *fed up*. Interestingly, this originated as a term in falconry—when you overfeed a hawk, it sulks until it has finished digesting; it has had enough food and is described as a *fed-up falcon*. The many people in the organization may have

- Conflicting communication and learning styles—they may not understand each other and the director;
- Different needs from their jobs—some will want to follow a process, some will want to "fly free";
- Jobs that are too stressful, too boring, or just badly designed—some people will want less stressful job, some will work best under pressure;
- All the fulfillment they need at a particular level, so reward and recognition at that level may not work—they may prefer a sabbatical to a pay raise, for example;
- Become irritated by what they see as "gestures" by the director, when they see other causes for their problems.

Motivation is not simple. People are not simple. Each person is an individual, with likes, dislikes, preferences, moods, and a life outside the team.

In this section the Internet/Web search words are: motivation, Maslow, hierarchy of needs, Motivation Improvement Programme, Mayo, Hawthorne, McGregor on Theory X and Theory Y, Likert on Exploitative, Benevolent, Consultative and Participative management of organizations, McClelland on achievement motivation, Argyris on bureaucratic, pyramidal versus humanistic, and democratic organizations.

A.2.5 Transactional analysis

We discussed transactional analysis in Chapter 4, based on Wagner's work [25], which itself refers to earlier work by Berne, for example, [26]. Wagner's book is useful to us here because it concentrates on work relationships rather than personal relationships or family situations.

The idea of transactional analysis as described by Wagner is that we are all six people rather than just one and that some of these six "inner people" are effective in dealing with others, but some of the "inner people" are not so useful. The six inner people or *ego states* Wagner identifies are:

- *The natural child*—an effective ego state that acts spontaneously, expresses feelings, and has need for recognition, structure and stimulation.
- *The adult*—an effective ego state that is logical and reasonable; it deals in facts rather than feelings.
- *The nurturing parent*—an effective ego state that is firm with others, but also understanding, sensitive, and caring.
- *The critical parent*—an ineffective ego state that uses body language, gesture and tone of voice to "tell others off," perhaps by sarcasm, pointing the finger, or a raised voice.
- *The rebellious child*—an ineffective ego state that gets angry and stays angry, is very negative, does not listen, may deliberately forget things, or procrastinate.
- *The compliant child*—an ineffective ego state that blames itself, uses a soft voice and whines, and is very careful and self-protective.

As we saw in Chapter 4, communication between the effective ego states is generally useful. We can create and play if we are both in "natural child." We can exchange facts clearly in adult to adult communication, and our nurturing parent ego states mean we can empathize and help each other. If we cross into the ineffective ego states, we will argue, whine, and blame without communicating or changing anything; in fact, we may make things worse. We need a way to move out of the ineffective states into effective states, and the de Bono Six Hats [5–7] that we discussed in this section allow us to do that, by providing a formal behavior pattern that acknowledges feelings as well as facts.

As Wagner discusses [25], the use of TA is not just in personal relationships in the family. Looking at our own behavior whether as a manager or in reacting to our managers and colleagues is useful, as supervising, delegating, making decisions, resolving conflict, and hiring or firing people may allcause unexpected behavior, and this may be caused by ineffective ego states.

A related study area is emotional intelligence. You will find checklists on these "soft skill" areas on the Chartered Management Institute Web site.

In this section the Internet/Web search words are: transactional analysis, "games people play," scripts, ego states, Berne, Wagner, and emotional intelligence.

A.3 Techniques to identify and classify problems and assess ideas for solutions

A.3.1 Cause–effect, root cause, and solution analysis

Cause–effect analysis helps us look for the root causes of problems, using *cause and effect diagrams*, sometimes called *fishbone diagrams* because of their shape, or *Ishikawa diagrams* after the person who first developed them, but now widely used in identifying quality and other problems [27, 28]. The diagram is simply a means of helping us think about and classify the causes of problems and their underlying root causes.

The first step is to draw a fishbone diagram with the effect (which is the problem or symptom noticed) on the head of the fish. Then, label the ribs of the fishbone to show typical areas of root cause. You can choose your labels. Typical ones are the 4 Ms—Manpower, Machines, Materials, Methods [27], or the "PEMPEM" labels, which gives People, Environment, Methods, Plant, Equipment, Materials [28], or you could make up your own labels. Figure A.1 shows a fishbone at the start of a meeting, as it was drawn up on the whiteboard. There had been a series of complaints about the help-desk provision. These have been divided into three groups, and each group placed on a separate arrow. A fourth arrow has been added for "other complaints" in case we have forgotten anything.

If you were in the team discussing this, you would now use discussion and brainstorming to generate causes for the effect. Try to group the ideas on the fishbone to show how they relate to each other. The next step is to allow the ideas to incubate. Robson [28] suggests putting the diagram on a public board so other people can see, comment, and add to it, and so that the people can suggest other causes. We discussed this example in Chapter 12, so you will see the completed fishbone in Figure 12.3.

Having identified some root causes, we may want to "reverse" the fishbone, to go from a problem to a solution [27, 28]. To do this, we draw the diagram in reverse, write our proposed solution in the box, and then under our fishbone headings discuss whether the proposed solution will work:

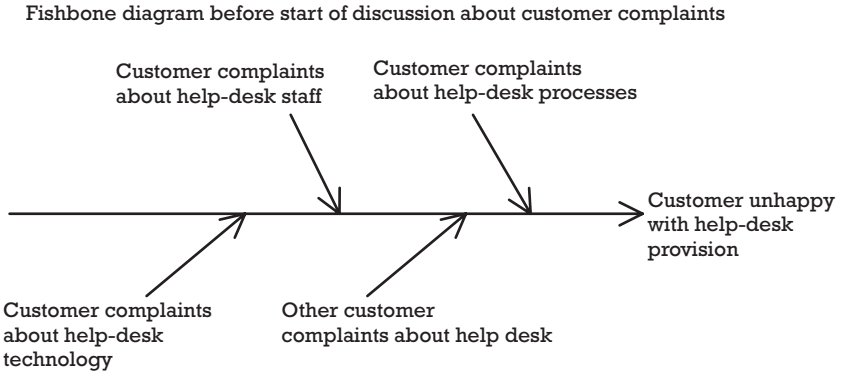


Figure A.1 Ishikawa fishbone diagram—empty at start of discussion.

- What actions do we require under the 4 Ms or PEMPEM or our own titles to make sure the solution works? (Note: whichever titles you use, make sure you have a place to look at time, money and staffing which will all constrain the solution.)
- What advantages or positive effects will the solution have?
- What disadvantages or negatives can we identify?

TQMI [27] suggest using different color pens for the advantages and disadvantages, so you could match pen colors to the de Bono Six Hat's colors, using yellow for optimistic and black for pessimistic views (see Section A.1.2). Figure A.2 shows an example solution analysis fishbone. We can see here that the initial thought is that a new IT system for the help-desk staff might solve the problems. However, the constraints of time, budget, and staffing, as well as the observation that a new system will only solve some of the problem, leads to rejection of a new IT system. Instead, the solution analysis will continue by looking at possible staff training and process changes to improve the service to customers.

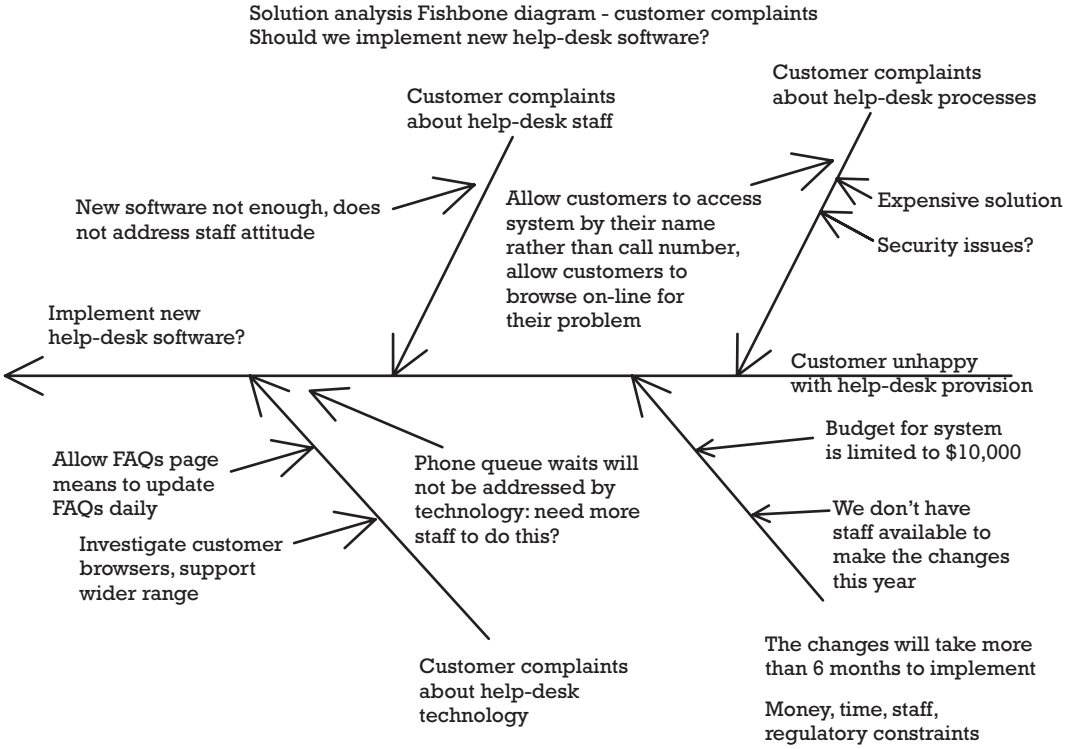


Figure A.2 Example of a solution analysis fishbone.

In these methods, brainstorming is a useful technique for gathering ideas. When using brainstorming, it is worth marking the start of the brainstorm session by agreeing or reminding ourselves of some rules [27, 28]. The precise rules may vary between organizations, so it is worth clarifying them. Typical rules include:

- All ideas are allowed.
- All ideas are documented.
- There is no discussion or evaluation during ideas gathering.
- Be relaxed.
- Do not criticize other's ideas.

In this section the Internet/Web search words are: Ishikawa fishbone, root cause analysis, solution analysis, and brainstorm.

A.3.2 Prototyping and ideas modeling

Prototyping is not an SDLC. It is a method of exploring a problem and potential solutions in order that the right solution can be selected. Prototyping is not just used in software; it is used in many fields to test design ideas. For example, if you make clothes, you may choose to use ready-made patterns as bought, or to alter ready-made patterns to fit your particular shape. Alternatively, you might build clothes against a general design but measured for a specific person (for example, a custom-made suit), or you could design your own clothes with unique patterns.

In all cases, you can use a prototype, called a *muslin*, *toile,* or *form*, which you use to try out the pattern. Using a cheap material, such as muslin, you build the garment from the pattern, in order to see if it turns out as you expect. Does the garment fit? Does it fall well? Does it flatter you? We do this to check our ideas and ability to understand what we need to do without cutting into the expensive cloth that we will use for the final garment.

Particularly if we are designing clothes rather than using an off-the-shelf pattern, a prototype is essential to try out new ideas.

Keen sewers will use a muslin to check out a new pattern; they are using the prototype to try out their understanding of *and adjustments to* "off-the-shelf" solutions. *You only use the prototype to try out ideas—you will not be wearing it in reality.*

If you want to see a good example of prototyping using muslin, look at a sewing publication, for example, *Threads* magazine [29], where user testing of sewing pattern software is done by making a set of prototype patterns for different sized and shaped people. The authors built prototypes to explore commercial off-the-shelf pattern fitting programs and decide which one to recommend.

In software, we can use prototypes in the same way. We can try out ideas if we are not sure what we want. In that situation, we do not want to build the real software. Instead we want to build a model of it, in order that we can explore the problem and potential solutions.

There are two types of prototyping in software, "lo-fi" and "hi-fi" [30, 31]. In hi-fi (high-fidelity) prototyping, we use software to build screens like those the customer will see. This can lead to people believing that the prototype is an almost complete system. "If the screens are ready," managers and customers may argue, "what else can be going on in the system that is so complicated? Why can't we have the system now?" But, as the builders, measurers, and supporters will tell you, the guts of the system are where the complexity lies. This is a big problem for software development and support. Think of our muslin; we were not expecting to wear it, so it is tacked together relatively roughly and not in a material we were expecting to wear or clean or alter; the muslin is not worn but thrown away. The same thing will be true of the software; are you really expecting to use it once and throw it away? I thought not; you will want to maintain and change it, so the rough and ready prototype, however polished on the surface, cannot be the system we deliver. An exception to this is the idea of evolutionary prototyping [31], which means that even the early prototypes must be built to be supportable and maintainable.

For this reason, many people prefer lo-fi (low-fidelity) prototyping. Here we do not build a prototype in software. We might use paper and pen, or sticky paper, or a presentation, or a whiteboard. We can still discuss the prototype and review it, but there is no danger of believing it is the software.

As we saw in Chapter 9, in Figure 9.3, whether we use the hi-fi or lo-fi approach or both, prototyping involves generating and discussing ideas and building a succession of models. As we do this, we gather good ideas and discard bad ideas. Eventually, we can decide what we want to do: develop some software, have a manual solution, or do nothing.

There are a number of related modeling and picturing techniques that can be worth exploring with lo-fi prototyping. These include Rich Picturing [32, 33] and Mind Mapping [34, 35]; both are ways of capturing ideas in an easily communicated and condensed way. In Graham Freeburn's workshop on the final day of EuroSTAR 2002 [35], for example, he summarized "What we learned at the conference" (3 days), with the participating audience contributing to one mind map on one slide, in 45 minutes. Sketching a picture of a problem, solution, or idea can clarify it. Using stories, metaphors, and analogy also helps in understanding problems, and this is why they are used in newer IT analysis methods such as UML [36], and there is no reason why they should not also be useful in more traditional approaches such as SSADM.

In this section the Internet/Web search words are: prototyping, rich picturing, mind mapping, use cases, and Buzan.

A.3.3 Assessing whether an idea is worth pursuing

Not all the ideas we generate are worth pursuing. We need to decide which are possible and cost-effective. Pareto analysis, risk analysis, and cost–benefit analysis are some of the techniques that help us to decide.

Pareto analysis [27, 28] is based on the "Pareto principle," which suggests that 80% of the problems are the result of 20% of the causes. This means that if we can identify solutions to those 20% of the problems, we should get a better, more effective payback from our chosen solution. To decide which problems to solve, identify which are the most frequent. To do this, gather data on the frequency of problems and the underlying causes. The most frequent problems and causes are candidates for resolving, so the ideas that address those problems may be worth pursuing.

In risk analysis, we use techniques such as brainstorming to list all the possible risks people can identify. Risks are sorted into groups, by topic area. We may find we have identified some issues (things that are problems now) and constraints (known limits to any solution we suggest) as well as risks. Risks have not yet happened, but may happen (turn into an issue or problem) in the future. When we assess a risk we need to know its likelihood of turning into a problem, as well as the impact if it does. There is therefore an element of forecasting in risk assessment. When we are assessing possible solutions to problems for their risk, we are asking whether the risk of pursuing a solution is greater or less than the risk of an alternative solution or doing nothing:

- If we do nothing, what could go wrong?
- If we adopt this solution and it goes as we expect what chain-reaction problems might we have?
- If we adopt this solution and it goes wrong, what problems arise from wrong delivery or nondelivery?

In each case we need to:

- List potential problems (risks).
- Identify the likelihood of each risk becoming a problem and use some scoring system.
- Identify the impact of each risk if it does become a problem and use some scoring system.
- Multiply impact by likelihood to get a risk score.
- Rank the risks.
- Identify what you can do to reduce risk or remove it.
- Decide whether the probable consequences of adopting or discarding the solution.

You will see variations on this theme in different people's work. For example, Hans Schaefer's work on risk assessment for software testing looks at the impact of failure as criticality of system, frequency of usage, and visibility of the problem [37]; whereas in TQMI's Failure Prevention Analysis [27], *probability* and *consequence* are used to calculate an overall rating for each possible failure's root cause, and these are used to rank to possible failures.

Cost–benefit analysis [28] is done by calculating the predicted benefits of the proposed change and setting this against the predicted cost. The steps in making a cost–benefit analysis, at its simplest are:

- Identify the financial costs of the solution—for example, direct costs of the solution in equipment, resources, and loss of time on revenue earning activities if staff are diverted to the project.
- Identify the nonfinancial costs of the solution. These may translate into financial costs but you may find it easier to start by listing them and then attempt to translate them. Examples might be adverse publicity, staff dissatisfaction with the change, and customer complaints.
- Identify the financial benefits of the solution—money saved by efficiency increases, additional sales, and customers.
- Identify the nonfinancial benefits of the solution. Again, these may translate into financial benefits but you may find it easier to list them first and put a price on them afterward. Examples might be improvements to staff morale, greater effectiveness in serving the customer, and improved image of the organization.
- Use this information to workout the "payback" over a number of years (see Table A.4). We can see that with the accumulated costs of the original implementation and support year by year, we will not get a payback until year 6.

When you look at the cost–benefit of an idea, you will need to look at how the benefits of the idea match to the organization's goals. This will give you an insight into whether the idea helps the organization's goals—a benefit—or whether it detracts from the organization's goals—a cost. You may well find that financial and nonfinancial targets have been set with the goals, and your cost–benefit discussion needs to reflect these targets. For example, if your organization uses a Balanced Scorecard, as we saw in Chapter 4, financial goals will be balanced against customer, process, and innovation costs and benefits. This information will help put together a cost–benefit discussion in terms familiar to senior management showing how their goals will be affected by your idea.

You may also wish to look at *earned value management* or *budgeted cost of work performed* measures [38], which allow you not only to track cost and budget, but also what the cost so far was supposed to achieve compared

Table A.4 Cost, Benefit, and Payback (in Dollars)

Year	*Cost This Year*	*Benefit This Year*	*Accumulated Cost*	*Accumulated Benefit*	*Benefit Minus Cost*
1	100,000	0	100,000	0	*−100,000*
2	2,000	25,000	102,000	25,000	*−77,000*
3	2,000	25,000	104,000	50,000	*−54,000*
4	2,000	25,000	106,000	75,000	*−31,000*
5	2,000	25,000	108,000	100,000	*−8,000*
6	2,000	25,000	110,000	125,000	15,000

with what actually has been done. To put it simply, we may have saved budget, but we have not delivered. In Table A.5, we see that although we are apparently under budget, we are behind in delivery, and therefore in earned value. We have, in fact, overspent compared with what we have delivered.

In this section the Internet/Web search words are: Pareto, 80–20 rule, failure prevention analysis, risk, impact, likelihood, probability, consequence, exposure, risk tree, risk-based testing, Schaefer, cost, benefit, payback period, financial planning, cash flow forecasting, profit and loss forecasting, business financial planning, balanced business scorecard, balanced scorecard, Kaplan and Norton scorecard, earned value management (EVM), and budgeted cost of work performed (BCWP).

A.4 Understanding aims and objectives

We saw in Chapter 9 (Figure 9.4) that we need to set aims, objectives, targets, and indicators. One useful technique is the Weaver triangle. This was originally developed by Jayne Weaver for use with nonprofit organizations, and, with her help, I then adapted it for use in IT and business projects [39] On a one-page diagram, the group identifies and agrees on the aim of the project (why it is being done) and associated indicators of success, then the objectives of the project (what is to be done) and associated targets. This helps identify where stakeholders have different aims for the project. The form is used to encourage teams to focus in one an overall aim or goal, and to show pictorially how the aims and objectives fit together. Some ground rules are:

- The aim should answer the questions "why are we doing this?" and "what difference will this make?"
- The specific aims should break down the overall aim in some detailed aims, but avoid having more than three to five of these, or you will get confused.
- Each specific aim also answers a "why?" and "what difference?" question.
- In order for the aims to be achieved, something needs to be done—these are the objectives—so each specific aim must be associated with at least one objective.
- The objectives each answer the question "What do we need to do in order to meet the aim?"

Table A.5 Earned Value Calculations

Budgeted Cost to Date	*Planned Delivery to Date*	*Actual Costs to Date*	*Actual Delivery to Date*	*Earned Value*
$1,000	Five documents (earned value $200 per document)	$800 (under budget)	Three documents (behind schedule)	$600 (this is $400 behind the expected earned value)

- There will be several objectives, which may be projects within a program or parts of a project, depending on their size.
- Each objective must be focused on achieving at least one of the aims, otherwise there is no in point doing it.
- Aims are measured by indicators that measure whether we are making the difference we intended.
- Objectives are measured by delivery targets such as savings, number of people affected, delivery dates, and budget.
- Indicator and target measures should be linked to measures used generally in the organization; for example, you could show how these measures link to the organization's balanced scorecard.
- Consensus is required between the stakeholders; this is not done by the managers and told to everyone else; it requires contributions and discussion from all the groups.

In this section the Internet/Web search words are: target, indicator, Gilb and Planguage, Seddon, and systems thinking, which will show alternative views about target setting and methods. There is none for Weaver triangle, as this is newly published.

A.5 Review techniques

There are five types of review [40]: management review, technical review, inspection, walkthrough, and audit, all of which are relevant during start-up, throughout the software-development life cycle (SDLC), and during delivery, as well as during the software-maintenance life cycles (SMLC) in the postdelivery period. Specialist testers regard them as a form of testing, because they are used to find and prevent defects in products and processes, but they can be used by any of the groups. Further, they may be used as an opportunity for communication between the groups. The review types have different purposes:

- The management review is carried out to check progress against plans.
- The technical or peer review has the purpose of identifying conformance to specification and finding defects in a document.
- An inspection is a formal review with the purpose of identifying and preventing defects, based on a sample from a document.
- A walkthrough is a review with the purpose of increasing understanding of a document.
- Audits are used to check process conformance rather than products.

We saw in Chapter 3 how useful taking part in reviews can be for the customer, and that the review process need not be complicated. Gilb and

Graham provide a full description of the Inspection process, with example forms [41], and the standard [40] provides an overview of each of the review processes. To get the best from reviews, it is best for people to have some training to understand how to review, and to have a process to follow. For all the reviews, it is good to:

- Set a policy for reviews. Decide what must be reviewed and which type of review is needed. Also decide what should be reviewed but could be left in an emergency, what could be reviewed if there is time, and what will not be reviewed, perhaps based on the risks associated with errors in the products or processes under review.
- Plan to have reviews. Allow time and resource for the activities as we saw in Chapter 4, and use a project review and audit plan structure like the one in Appendix B.
- Plan each particular review—its goals, who should be involved, and what specifically you want them to check, write, or improve the preparation checklists. An example of an audit checklist is in Appendix B.
- Communicate with everyone who will take part to explain what will happen and why the review will be done.
- Train reviewers so they know *how* to prepare—it is not enough to just read a document. Try to use it, or to match it against a related document such as standards, policy, or another product document.
- Ensure that each reviewer has prepared properly—do not hold the review unless everyone has prepared. A review where a reviewer is ill-prepared is just a waste of time and money because you will not find so many defects and you will not gain the same understanding.
- Make sure the review is of the product or process *not the person*—it needs to be objective and helpful not accusing. Some of the communication techniques in this appendix may help.
- Follow up the review. Improve this product and this iteration of the process, but also put in place improvements for the future to prevent future problems.
- Measure the reviews cost–benefit (time/money spent and faults removed/failures prevented) ratio and suggest improvements to the efficiency and effectiveness of the review process itself.

In this section the Internet/Web search words are: software review, technical review, inspection, and Fagan inspection statistical process control.

A.6 Improving graphics in reporting

I thoroughly recommend Edward Tufte's books [42–44]. Not only are they packed with useful information, but they are also a delight to handle and to read, being their own example of good information design. In Chapter 4, I

showed an example of a distorted graphic (Figure 4.7) and the equations proposed by Tufte that allow us to measure the distortion of data in a graphic [42].

The *lie factor*, which we saw shown in Chapter 4, Figure 4.7, gives a measure of the exaggeration of data in a graphic. The *data ink ratio,* shown in Figure A.3, is a measure of how much of the graphic provides information and how much is decorative. The data density, Figure A.4, compares the number of data items with the size of the graphic. In summary, the equations are as shown in Figure A.5.

In this section the Internet/Web search words are: Tufte, information design, and graphics press.

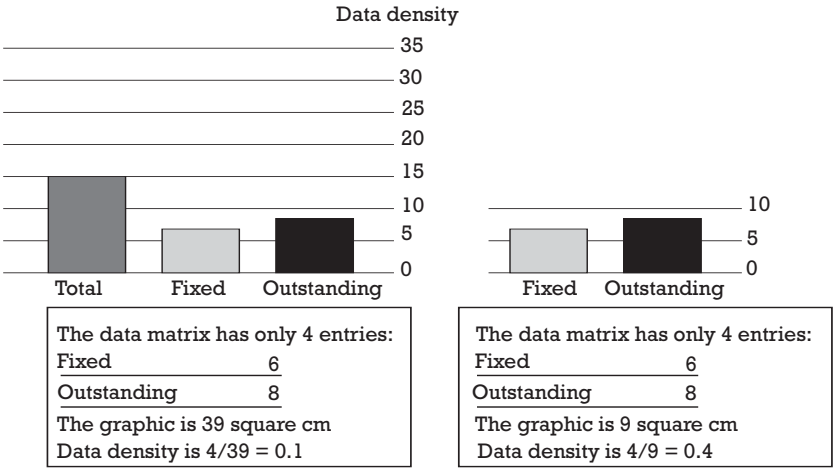


Figure A.3 Data ink ratio in graphics. (*After:* [42].)

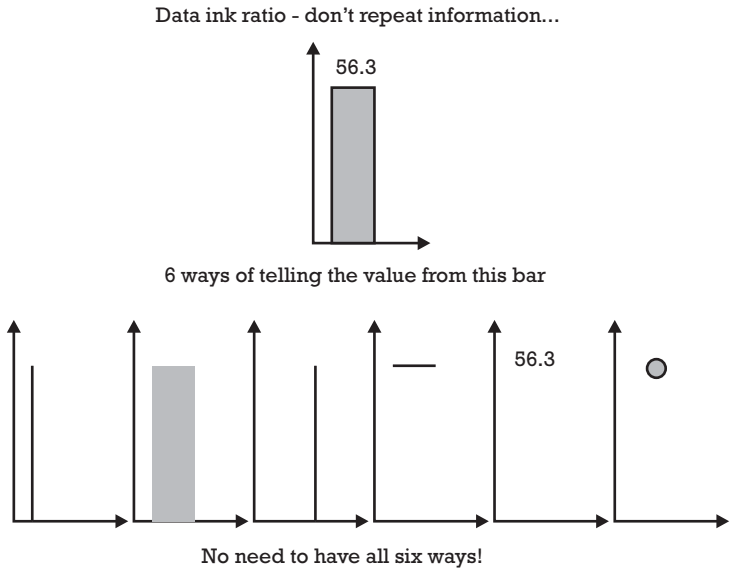


Figure A.4 Data density in graphics. (*After:* [42].)

$$\text{The Lie Factor} = \frac{\text{size of effect shown in graphic}}{\text{size of effect in data}}$$

$$\text{Data ink ratio} = \frac{\text{data ink}}{\text{total ink used to print the graphic}}$$

$$\text{Data density} = \frac{\text{number of entries in data matrix}}{\text{area of data graphic}}$$

Figure A.5 Tufte's data graphics equations (*From:* [42]. © 1983 Graphics Press. Reprinted with permission.)

A.7 Useful sources and groups

This section is just a starting list of useful further information and groups. There are many quality techniques and standards that I have not covered in this book. New ideas for improving IT provision are always coming to light. The list in Table A.6 and the following Internet/Web search words are just a starting point for your future research.

Table A.6 Some Sources for QA and QC Standards, Frameworks, and Best Practice

The National Strategy to Secure Cyberspace	http://www.whitehouse.gov/pcipb/
IT Governance Institute	http://www.itgi.org
Information Systems Audit and Control Association	http://www.isaca.org
The Institute of Internal Auditors	http://www.theiia.org/
Institute of Quality Assurance	http://www.iqa.org/
American Society for Quality	http://www.asq.org
Testing Standards Working Party	http://www.testingstandards.co.uk
Sticky Minds	http://www.stickyminds.com
BSI	http://www.bsi.org.uk
ISO	http://www.iso.ch
IEEE	http://www.ieee.org
TickIT	http://www.tickit.org/international.htm
National Institute Standards	http://hissa.ncsl.nist.gov
Office of Government Commerce	http://www.ogc.gov.uk
Successful IT: Modernizing Government in Action	http://www.ogc.gov.uk/index.asp?docid=2632
Six Sigma and robust design	http://www.isixsigma.com
Juran articles	http://www.juran.com
Tufte articles	http://www.edwardtufte.com
Deming articles	http://www.deming.org/
Acronyms expanded	http://www.acronymfinder.com/
Technical terms explained	http://whatis.com
EFQM—European Foundation for Quality Management	http://www.efqm.org
BQF—British Quality Foundation	http://www.quality-foundation.co.uk
SEI—Software Engineering Institute	http://www.sei.cmu.edu
ESI—European Software Institute	http://www.esi.es/
FORTEST—formal methods and testing	http://www.fortest.org.uk/
BCS Industry Structure Model	http://www.bcs.org.uk
BCS Special Interest Groups	http://www.bcs.org.uk
BSC Qualifications	http://www.bcs.org.uk
International Software Testing Qualification Board	http://www.istqb.org
Chartered Management Institute	http://www.managers.org.uk
For an alternative view of quality models…	http://www.lean-service.com
Balanced Score Card Group	http://www.bscol.com
Software Quality Professional	Journal
Better Software (previously STQE)	Journal
Project Manager Today	Journal

In this section the Internet/Web search words are: quality, excellence, quality improvement, governance, audit, standards, software engineering, Crosby, Juran, Deming, CMM®, TMM®, process improvement, patterns, exploratory testing, agile methods, agile manifesto, key process, software testing, verification, validation, continuous improvement, Just In Time, Six Sigma, Taguchi, and Sarbanes-Oxley Act.

References

[1] Busco, C., et al., "When Crisis Arises and the Need for Change Confronts Individuals: Trust for Accounting and Accounting for Trust," http://www.cimaglobal.com/downloads/research_enroac_busco.pdf, November 2003.

[2] Belbin, R. M., *Management Teams—Why They Succeed or Fail*, London, England: Butterworth Heinemann, 1981.

[3] Belbin, R. M., *Team Roles at Work*, London, England: Butterworth Heinemann, 1995.

[4] Belbin Associates, "Belbin Team Roles," http://www.belbin.com/belbin-team-roles.htm, October 2003.

[5] de Bono, E., *Six Thinking Hats®*, New York: Penguin Books, 1999.

[6] Pas, J., "Emotional Intelligence as the Key to Software Quality," *EuroSTAR Conference*, Stockholm, Sweden, 2001.

[7] de Bono, E., "Edward de Bono's Web," http://www.edwdebono.com/, October 2003.

[8] Kroeger, O., J. M. Thuesen, and H. Rutledge, *Type Talk at Work: How the 16 Personality Types Determine Your Success on the Job,* New York: Bantam Doubleday Dell, 2002.

[9] Team Technology Web site, "Working Out Your Myers Briggs Type," http://www.teamtechnology.co.uk/tt/t-articl/mb-simpl.htm, October 2003.

[10] Team Technology Web site,"The Mother of Strategic Systems Issues: Personality," http://www.teamtechnology.co.uk/tt/t-articl/news1.htm, October 2003.

[11] Mullins, L. J., *Management and Organisational Behaviour*, 5th ed., New York: Financial Times/Pitman, 1999, p. 313.

[12] Honey, P., and A. Mumford, *The Learning Styles Helper's Guide*, Maidenhead, England: Peter Honey Publications, 2002, http://www.peterhoney.com. PeterHoney.com, 10 Linden Avenue, Maidenhead, Berks, SL6 6HB. Tel.: 01628633946. Fax: 01628633262. E-mail: info@peterhoney.com.

[13] Honey, P., "Learning Styles," http://www.peterhoney.co.uk/product/learningstyles, October 2003. PeterHoney.com, 10 Linden Avenue, Maidenhead, Berks, SL6 6HB. Tel.: 01628633946. Fax: 01628633262. E-mail: info@peterhoney.com.

[14] Campaign for Learning, http://www.campaign-for-learning.org.uk/aboutyourlearning/whatlearning.htm, October 2003.

[15] Hicks, L., "The Nature of Learning," in L. J. Mullins, (ed.), *Management and Organisational Behaviour*, 5th ed., New York: Financial Times/Pitman, 1999, pp. 344–375.

[16] McHale, J., "Innovators Rule OK—Or Do They?" *Training & Development,* October 1986, http://www.kaicentre.com/.

[17] Kirton, "Adaptors and Innovators Defined," KAI Web site, http://www.kaicentre.com/, July 2003.

[18] Warden, R., and I. Nicholson, *The MIP Report—Volume 2—1996 Motivational Survey of IT Staff*, 2nd ed., Bredon, England: Software Futures Ltd., 1996.

[19] Hackman, J. R., and G. R. Oldham *The Job Diagnostic Survey: An Instrument for the Diagnosis of Jobs and the Evaluation of Job Redesign Projects*, Technical Report No. 4, New Haven, CT: Yale University, Department of Administrative Sciences, 1974.

[20] Accel-team.com, http://www.accel-team.com/human_relations, October 2003.

[21] Maslow, A., *Motivation and Personality*, New York: Harper and Row, 1954.

[22] Maslow, A., *Motivation and Personality*, New York: Harper and Row, 1970.

[23] Gywnne, R., "Maslow's Hierarchy of Needs," http://web.utk.edu/~gwynne/maslow.HTM, November 2003.

[24] Accel-team.com, "Maslow's Hierarchy of Needs," http://www.accel-team.com/human_relations/hrels_02_maslow.html, October 2003.

[25] Wagner, A., *The Transactional Manager—How to Solve People Problems with Transactional Analysis*, Denver, CO: T.A. Communications, 1981.

[26] Berne, E., *Games People Play*, New York: Grove Press, 1964.

[27] TQMI, *Problem Solving—Tools and Techniques*, Frodsham, England: TQMI, 2001.

[28] Robson, M., *Problem Solving in Groups*, Aldershot, England: Gower, 1995.

[29] Neukam, J., and J. Sauer, "Pattern-Drafting Software," *Threads*, May 2003, pp. 42–49.

[30] Hohmann, L., "Lo-Fi GUI Design," *Software Testing and Quality Engineering*, Vol. 1, No. 5, September 1999, pp. 24–29.

[31] Nance, R. E., and J. D. Arthur, *Managing Software Quality*, New York: Springer-Verlag, 2002.

[32] "Drawing Concerns: A Structured Rich Picturing Approach," http://business.unisa.edu.au/cobar/documents/richpic_colin.pdf, November 2003.

[33] Rose, J., "Soft Systems Methodology as a Social Science Research Tool," http://www.cs.auc.dk/~jeremy/pdf%20files/SSM.pdf.

[34] Buzan, T., *The Mind Map Book*, London, England: BBC Consumer Publishing, 2003.

[35] Freeburn, G., "Mind Mapping 101 for Testers," *EuroSTAR Conference*, Edinburgh, Scotland, 2002.

[36] Fowler, M., and K. Scott, *UML Distilled*, Reading, MA: Addison-Wesley, 1997.

[37] Schaefer, H., "Testing—The Bad Game and the Good Game," *BCS SIGiST Conference,* Edinburgh, Scotland, 1997.

[38] Pavyer, E., "An Introduction to Earned Value Management," *Project Manager Today*, Vol. 11, April 2003.

[39] Evans, I., "The Troubled Project—Best Practice from Theory to Reality," *EuroSTAR Conference*, Stockholm, Sweden, 2001.

[40] IEEE 1028™ Standard for Software Reviews, 1997.

[41] Gilb, T., and D. Graham, *Software Inspection,* Reading, MA: Addison-Wesley, 1993.

[42] Tufte, E., *The Visual Display of Quantitative Information* (Equations from p. 57, the Lie Factor; p. 93, The Data Ink Ratio; p. 162, Data Density), Cheshire, CT: Graphics Press, 1983.

[43] Tufte, E., *Visual Explanations*, Cheshire, CT: Graphics Press, 1990.

[44] Tufte, E., *Envisioning Information*, Cheshire, CT: Graphics Press, 1997.

Selected bibliography

Crosby, P., *Quality Is Free,* New York: Penguin, 1980.

Garfinkel, H., *Studies in Ethnomethodology*, Englewood Cliffs, NJ: Prentice-Hall, 1967.

Handy, C., *Understanding Organizations*, New York: Penguin, 1993.

Hofstede, G. H., *The Game of Budget Control*, London, England: Tavistock, 1968.

Humphreys, W., "Pathways to Process Maturity: The Personal Software Process and Team Software Process," http://interactive.sei.cmu.edu/Features/1999/June/Background/Background.jun99.htm, August 2003.

Northcott, D., *Capital Investment Decision Making*, San Diego, CA: Academic Press, 1992.

Ould, M., *Managing Software Quality and Business Risk*, New York: Wiley, 1999.

Perry, W. E. and R. W. Rice, *Surviving the Top Ten Challenges of Software Testing: A People-Oriented Approach*, New York: Dorset House, 1997.

Winant, B., "Visual Requirements," *STQE,* June 2003, pp. 34–42.

Appendix B

Quality Planning Documents and Templates

B.1 The document family

As we saw in Chapter 4 (see Figure 4.1), the quality planning required for a particular project will be based on the organization's standards, including its policies and chosen methods of work. Within an individual project, once the aims, objectives, risks, and constraints for the project are understood, we can develop a family of documents which describe how *for this project* we will carry out our work. We have seen that quality planning for the SDLC should involve all the groups. This activity should be part of project planning. There are a number of documents that might be needed; on a small project I would have these as paragraph headings in the project plan, whereas on a large project, you might need a series of documents, with levels of detail from a strategy through detailed plans. Table B.1 lists the levels of quality planning documents.

Table B.1 Quality Planning Documents

This is what we do as an organization	Policy documents organization level, brief, part of organization QMS
This is how we will do it as an organization	Standards and processes organization level, descriptive, steps "how to", organization Quality Management System (QMS) for all activities, not just QA/QC activities
This is what we will do for this SDLC	Project QMS or quality plan
	Program/project level, documents tailoring decisions based on risk and constraints, choice and tailoring of SDLC (see Chapter 10), may include policy, standards, and processes, where these differ from the organization level documents, for QA, QC, management, and build activities
This is how we carry out QA/QC activities for this SDLC	QA: Audit and review strategy and overall plan
	QC: Document review strategy and overall plan
	QC: Testing strategy and overall plan
	Project/program level, responsibilities, overall approach based on tailored QMS, which groups will contribute to which QA/QC activities
This is how we will carry out this particular QA/QC activity	Detailed plans for each particular audit, review, document review, test level, stage within SDLC, specific, detailed approach, detailed task plan, specific responsibilities

In an organization with a documented Quality Management System (QMS) there will be a documented policy for both QA and QC activities, describing at a high level what is expected for any SDLC. Even if the organization does not have written policies, there will be unwritten, taken-for-granted assumptions: "We always do it this way." The policies will be backed up by process descriptions, procedures, and standards. Again, in some organizations these will be documented, but in others they are based on word of mouth, individual expertise, and training. Some important questions to think about are:

- Have we chosen the appropriate standards/methods/tools and techniques to carry out our work?
- Do they need adapting for this project?
- Are there specific expectations or rules from our customers about how we do the work—do we need to follow a particular standard or method?
- Are we doing the right QA/QC activities bearing in mind the risks and constraints?
- Are the QA and QC activities sufficient to meet any external or internal requirements?
- Does everyone agree that these QA and QC activities are needed?
- Have they been defined in a way that will allow measurement of product and process?

In order to allow these questions to be considered, I think it is useful to have a written QMS that contains the policies, processes, and standards, but, importantly, has a rule that "the processes and standards can be tailored to suit the risks and constraints for the particular project" We have seen some examples of tailoring in Chapter 12. In Figure B.1, we can see an example document family. Remember this is just an example; you may choose to have fewer documents by combining some of them.

We see in the figure that the documents we developed during start-up (Chapter 9) plus the company policies and standard drive what appears in the project documents. The documents in the project that address quality include the project plan itself, the risk-management plan, the configuration-management plan, and the quality plan. These documents—or project plan parts in a small project—need to be developed together; they are a complementary set. Each may be the parent for a document family. In this figure, we will develop the quality branch. The quality plan divides into three main parts in this project: an audit plan, a plan for which documents to inspect or review, and an overall plan for testing. Each of these branches has a similar structure, so let us just look at the audit branch. There will be several audits, and each will need a plan. This is not a large document; it just means that we need to agree on the time and place for each audit, who will be involved, book rooms and so on. It might be just a checklist or an e-mail to confirm

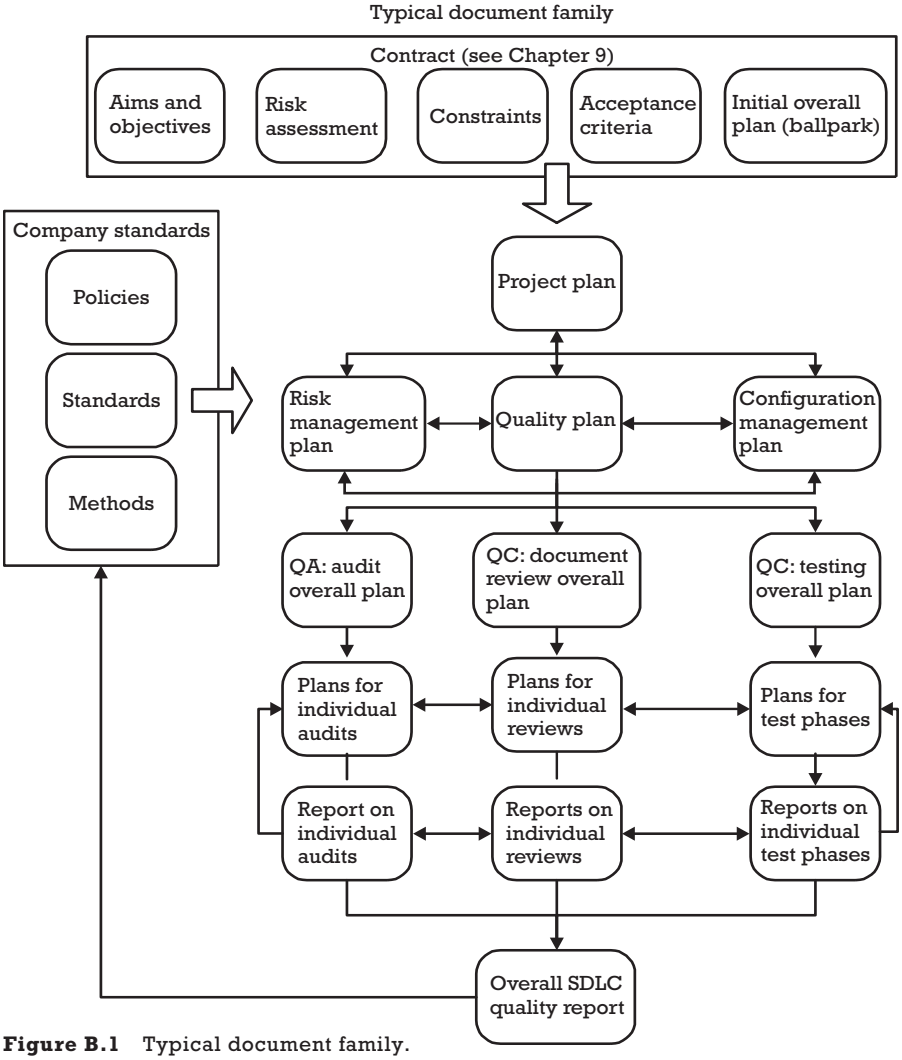


Figure B.1 Typical document family.

what the auditor and team being audited have agreed. After each audit, a short report is needed, just the good points and areas for improvement in a list, with priorities. Do not write much, but do share it with the team you have audited and get their feedback. Finally, all the quality activities feed into an overall quality report, which helps all the groups decide on the outcome of the project, but also feeds improvements to the company standards.

B.2 Why we use document templates

We need to know which documents we have to write, why we write them, and what their contents should be. To save everyone time deciding what is needed, it is useful to have a set of standard document templates. There are a number of standards that provide document templates, for example:

- IEEE standards, including standards for project plans [1], quality plans [2], test documentation [3], requirements documents [4], and user documentation [5];
- BSI standards, including standards for documenting component testing [6].

The standard bodies update these documents, and so it is best to consult the standards bodies for the latest versions. For example, IEEE 829™ [3] is the Test Documentation standard. If you look on the IEEE Web site [7], you will see IEEE 829:1983, which is superseded, and IEEE 829:1998, which at time of writing is the current standard, but you will also see IEEE P829, which is a project to revise the standard by bringing it into line with current thinking, and other, more recent standards, which will result in the 1998 version being superseded.

These standards should be your source for deciding on the content of your own document templates, but almost inevitably, you will find that you want to adapt the standards in some way. Tables B.2 through B.5 are deliberately not a complete set of documents, so that you are encouraged to find out what the standards bodies are offering, and to obtain the latest standards for each type of document. Sources of information are:

- IEEE [7];
- BSI [8], including the BSI Software and Systems Quality Framework (SSQF) [9];

Table B.2 Example of a Project Plan Template

Project plan	Not just the schedule. To allow discussion of whether the project is possible—using tools and techniques like PERT (see Chapter 4), we can model the project. Loosely based on [1], tailored over use.			
Project plan for project:	Author	Date	Status	Version
Reference	To higher level documents—do not repeat information from higher level documents in this document; just document differences, new information, and exceptions (e.g., project authorization, terms of reference)			
Introduction	Overview			
Deliverables	List of all the deliverables			
Evolution	Change control			
Vocabulary	Definitions and acronyms on this project			
References	Documents that will expand on the information in this document if required (e.g., quality plan, configuration management plan, risk management plan)			
Organization	Process model, organizational structure, organization boundaries and interfaces, project responsibilities, managerial process, objectives, and priorities			
Risk	Assumptions, dependencies and constraints, risk management, monitoring and controlling mechanisms			
People	Staffing plan, staffing needs, staff available, training and induction plans, rewards and recognition plans, teams, team dynamics			
Methods	Technical processes, methods, tools and techniques, software documentation, project support functions			
Schedule	Work packages, dependencies, resource requirements, budget and resource allocation, schedule			

Table B.3 Example of a Quality Plan Template

Quality plan	Document approach, specifically exceptions to policy and standards, help you consider and plan for QA/QC activities, evidence of decisions, rule book for actions, checklist for QA/QC activities. Loosely based on [2], tailored over experience.			
Quality plan for project:	Author	Date	Status	Version
Reference	To higher-level documents—do not repeat information from higher-level documents in this document; just document differences, new information and exceptions (e.g., project authorization, terms of reference, project plan)			
Quality objectives	Summary, list of items subject to the quality plan, features of interest, items not subject to the quality plan, specific quality objectives			
Approach	General approach, selection of methods including QA/QC Methods, pass and fail criteria, sign off procedures, suspension and resumption, corrective actions			
Exceptions	List of differences from policies and standards, adaptations made to templates, other changes to normal process			
QM tasks and deliverables	QM deliverables, QM, QP, QA, QC tasks, test, inspection, examination and audit programs			
Scheduling/ resourcing	Environmental needs, responsibilities and authorities, resources required and resources available, staff and training needs, schedule, budget			
Risks	Specific risk areas, float available, contingency plans			
Review	Change control on this plan, review points on this plan			
Other	What else might we need to think about?			
Lower-level documents	List documents which will expand on this information if required (e.g., configuration management plan, risk management plan, test plan)			

Table B.4 Example of a Risk Management Plan Template

Risk management plan	Gain agreement from all parties on control of risk, define management processes for RM. This template is experience-based; there is an IEEE standard in [10].			
For Project:	Author	Date	Status	Version
Reference	To project plan, risk assessment, risk register			
Introduction	Purpose, Scope, Definitions/mnemonics			
Management	Responsibilities and authorities, escalation			
Activities	Risk reassessment cycles—timing, responsibilities Risk identification and assessment activities Risk containment measures—criteria for deciding whether a risk is to be treated by prevention, mitigation, or contingency plans, or is to be accepted Relationship to quality plan—QA and QC activities related to risk			
Review	Change control on this plan, review points on this plan			
Other information	What else might we need to think about?			
References	Quality plans, including audit plans, review plans, test plans			
Approvals	Who needs to approve? Who needs to buy in? Who needs to know?			

- ISO [11];
- Testing Standards Web site [12], including Reid's paper comparing testing standards [13];
- ITIL [14];
- itSMF [15].

Table B.5 Example of a Configuration Management Plan Template

Configuration management plan	Gain agreement from all parties on control of changes, define management processes for CM Loosely based on [16], tailored from experience.			
For Project:	Author	Date	Status	Version
Reference	To higher level documents—do not repeat information from higher level documents in this document; just document differences, new information and exceptions			
Introduction	Purpose Scope Definitions/mnemonics			
Management	Organizations, SCR responsibilities, Interface control, status accounting, audits, CC board			
Activities	Configuration identification, configuration control, configuration status accounting, audits and reviews, tools, techniques and methodologies, supplier control, records collection and retention			
Review	Change control on this plan, review points on this plan			
Other Information	What else might we need to think about?			
References	List documents which will expand on the information in this document if required			
Approvals	Who needs to approve? Who needs to buy in? Who needs to know?			

B.3 Using the document standards to provide your own templates

Put together a set of templates as a starting point for all projects, and have two or three "sizes"—perhaps an "emergency" template, an "agile" template, and a "high-risk" template, as we discussed in Chapter 10—and set the policy for projects by adapting these. Make sure these templates provide enough information for anyone who needs to understand or take on aspects of the work. For example, do the supporters need additional information because they will take on the maintenance of the system after delivery?

B.4 Auditing considerations

Consider whether you will be audited, who by, and if there is an expectation that you adhere to particular documentation standards. If so, you need to understand whether you are allowed to adapt the published standard; for example a customer may require you to provide "test documentation written to meet the IEEE 829 Test Documentation Standard" [3]. What does this mean? To which date standard are they referring? Can you adapt it? If you can adapt, document the adaptation and make it clear how it meets the standard.

B.5 The team's information needs

The team size and experience may affect how much you document. For a small project team, with membership from all the groups and good

communication skills, who have worked together before and who are working on a fast track project, you need to use these documents as checklist of things to remember. Maybe you only have to document the exceptions to rules you have previously developed. However, if you have a large team of people, and perhaps working together for the first time, perhaps from different organizations, you need to find a way to communicate and agree how things will be done. Try using the document headings as an agenda for a discussion meeting. Then document and review the outcome to check that you all have a common understanding. Remember to communicate the information; I remember seeing a very good test strategy that was only known to its author; no one else had read it, yet the development manager and the operations group both needed information from that document.

B.6 Adapting templates

If you adapt document templates for a particular project, when you start your next project, go back to the original templates. Otherwise you will forget why you adapted the template in a particular way, and perhaps miss something important from the original template. Expect the document templates to change over time. When you make changes to the templates, go back to the standards and think about *why* each section is there and whether you will be missing something if you change it (see Tables B.2 through B.5).

B.7 Keep it brief—do not repeat or copy information

Try to evolve a family of documents, with the lower-level documents only adding information, not repeating information from higher up the family tree. An example family tree is shown in Figure B.2. You will see that we may not need the detailed plans on all the branches of the tree.

As a general rule, just document any changes and exceptions. Discourage people from copying chunks from one document to another; repeated information makes update harder.

Do not allow people to pick up completed documents, copy them, and alter them; they may stop thinking about the content of the document and producing it may become a chore done for no purpose. The important thing about these documents is that they help us think and solve problems.

B.8 Do you need a document at all?

We write documents to communicate and to reach agreement. You may not need a set of text documents on paper; that may not be the best way to communicate or to reach agreement. You may be better off with a Web site, a presentation, a notice board, diagrams, a spreadsheet, a video of the discussion meeting, a video of the project sponsor—you decide what is best.

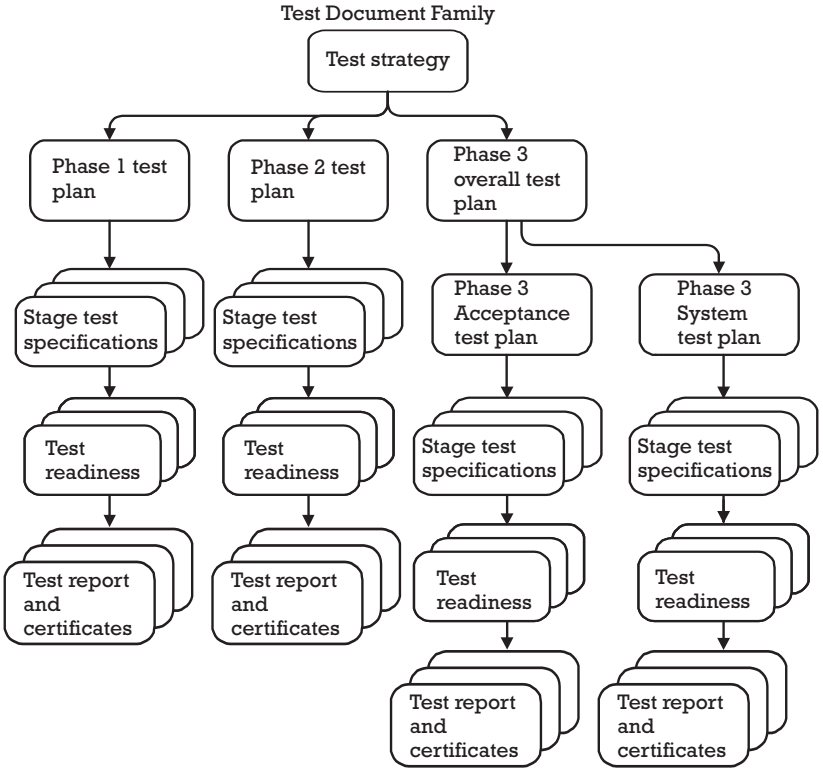


Figure B.2 Typical test documentation family. (*After:* [3].)

B.9 Simple project audit plan and report templates

These documents are ones I have developed based on the test and quality documentation [2, 3], and then tailored. For each planned audit, for example, in a spreadsheet, collect the following information, initially for planning and then for tracking:

- For *Planning* the headings are: Topic area, Topic detail, Audit planned month, Audit team, Audit place, Audit goal.
- For a *Detailed plan for one audit*: Audit checklist or reference to it.
- For *Tracking against plan* the headings are: Audit done date, Audit outcome, refer to Audit report, All issues resolved?

For a particular audit, the audit detailed plan may be a checklist. An example audit checklist that the auditor has started to develop for code maintainability might look like Table B.6.

The report for audits may be a text report, or it could be a spreadsheet of positive observations and areas of concern, or you may wish to log nonconformances to standards in the same way and place that you log inspection and test defects—in the defect logging system—or you might wish to use the audit primarily to identify risks, in which case log them in the risk register. For each point identify:

Table B.6 Example of an Audit Checklist (Maintainability Audit)

For Project:	Topic area	Audit team	Date	Status	Version
	Code quality—Maintainability				
Logistics and planning	Speak to team leader of the area to be audited and explaining what will happen Identify who will be interviewed Set times for a start meeting, interviews and a closing meeting Book rooms if required				
Review goal	Establish whether the project deliverables will be acceptably maintainable				
Initial Questions	What are the maintainability acceptance criteria for the project? Are these set at a level which meets the customer's needs? What analysis of maintainability is being carried out by the team? How complex is the code? How internally consistent and correct is the code? Does the code have areas that cannot be executed? How does the team check that data is defined and used correctly across interfaces? What static analysis tools being used to support maintainability measures? Which factors does the chosen tool address? How easily will changes be made to the delivered code?				
Metrics	Number of programs checked/number of programs For each program checked: Program size: Lines of code or number of programs or number of objects Code complexity: as measured using the ABC tool set Developer assessment (perception measure) of maintenance ease: predicted number of hours to make changes (see example of changes set), predicted number of faults, perception of "trickiness" Note: Check whether they are using ISO 9126 [17] for any Maintainability measures				
Method	Static analysis of code using tools and inspection Interview of project members Interview development manager Inspect coding standards				
Audit date	Refer to audit report				

- The specific point for example by audit name/number/date;
- Auditor name (person who raised the point);
- Review area;
- Reference to document or source of information;
- Description of area of concern or positive observation;
- Metric used and measurement;
- If this is an area of concern, identify risks associated (likelihood of problems arising and impact of those problems if they arise);
- If this is a reaudit, assessment of changes in risk since last audit;
- Prediction of outcome;
- Suggested resolution;
- Comment from audited team/team leader;

- Actual resolution;
- Date resolved
- Sign-off by audited team/team leader;
- Sign-off by auditor/audit team leader.

References

[1] IEEE™ 1058:1998 Standard for Software Project Management Plans.

[2] IEEE™ 730:1989 Standard for Software Quality Assurance Plans.

[3] IEEE™ 829:1998 Standard for Software Test Documentation.

[4] IEEE™ 830:1998 Recommended Practice for Software Requirements Specifications.

[5] IEEE™ 1063:2001 Standard for Software User Documentation.

[6] British Standards Institute, BS7925-2:1998 Software Testing, Part 2 Software component testing.

[7] IEEE Web site, http://www.ieee.org.

[8] British Standards Institute Web site, http://www.bsi.org.uk.

[9] British Standards Institute, PD0026:2003, "Software and Systems Quality Framework—A Guide to the Use of ISO/IEC and Other Standards for Understanding Quality in Software and Systems," May 2003.

[10] IEEE 1540-2001™ Standard for Software Life Cycle Processes—Risk Management.

[11] International Standards Organization Web site, http://www.iso.ch.

[12] Testing Standards Working Party Web site, http://www.testingstandards.co.uk.

[13] Reid, S. C., "Software Testing Standards—Do They Know What They Are Talking About?" http://www.testingstandards.co.uk/publications.htm, August 2003.

[14] IT Infrastructure Library, Web site http://www.itil.co.uk.

[15] IT Service Management Forum, http://www.itsmf.com, October 2003.

[16] IEEE 828:1998™ Standard for Software Configuration Management Plans.

[17] International Standards Organization/International Electrotechnical Commission (ISO/IEC) DTR 9126 Software Engineering—Software Product Quality (Parts 1–4, 2000/2001).

About the Author

Isabel Evans has 20 years of experience in the IT industry, mainly in quality management, testing, training, and documentation. She has helped organizations in the development of procedures, standards, and methods to aid in the testing of software during development and maintenance projects. She has managed test groups and performed testing design and development for the acceptance and system testing of packages and bespoke systems. Ms. Evans has also provided quality assurance support, release management, and customer support for IT organizations. She has worked independently since 1992, running her own company, IE Testing Consultancy Ltd. After working closely with Testing Solutions Group (TSG) since 2002, Ms. Evans joined the company in January 2004.

Ms. Evans writes and presents courses; for TSG, these include courses for beginners, the ISEB Foundation, and Practitioner Certificates in Software Testing and specialist courses in testing and quality methods. In the past she has provided training and tutorial material in quality management, project management, and documentation skills. While she has a sound theoretical basis for testing and quality, her own experience as a tester and as a quality consultant provides a practical approach and real-life experiences.

As well as presenting seminars and training courses to clients, Ms. Evans has spoken on software quality, testing, and test management at conferences in the United Kingdom, Europe, and the United States, including EuroSTAR, PSST, Quality Forum, BCS SIGIST, and the Year 2000 and EURO Summit. She regularly attends conferences, courses, and meetings in her interest areas. Ms. Evans has been a member of various working parties and groups to contribute to improvement in software quality and testing, including the Quality Forum Testing Metrics Forum, the Customer Satisfaction Measurement working party, and the BCS SIGIST Test Standards Working Party, currently developing nonfunctional testing standards.

Index

D

H

I

K

L

M

Q

R

S

T

Recent Titles in the Artech House Computing Library

Achieving Software Quality through Teamwork, Isabel Evans

Action Focused Assessment for Software Process Improvement, Tim Kasse

Advanced ANSI SQL Data Modeling and Structure Processing, Michael M. David

Advanced Database Technology and Design, Mario Piattini and Oscar Díaz, editors

Agent-Based Software Development, Michael Luck, Ronald Ashri, and Mark d'Inverno

Building Reliable Component-Based Software Systems, Ivica Crnkovic and Magnus Larsson, editors

Business Process Implementation for IT Professionals and Managers, Robert B. Walford

Data Modeling and Design for Today's Architectures, Angelo Bobak

Developing Secure Distributed Systems with CORBA, Ulrich Lang and Rudolf Schreiner

Discovering Real Business Requirements for Software Project Success, Robin F. Goldsmith

Future Codes: Essays in Advanced Computer Technology and the Law, Curtis E. A. Karnow

Global Distributed Applications with Windows® DNA, Enrique Madrona

A Guide to Software Configuration Management, Alexis Leon

Guide to Standards and Specifications for Designing Web Software, Stan Magee and Leonard L. Tripp

Implementing and Integrating Product Data Management and Software Configuration, Ivica Crnkovic, Ulf Asklund, and Annita Persson Dahlqvist

Internet Commerce Development, Craig Standing

Knowledge Management Strategy and Technology, Richard F. Bellaver and John M. Lusa, editors

Managing Computer Networks: A Case-Based Reasoning Approach, Lundy Lewis

Metadata Management for Information Control and Business Success, Guy Tozer

Multimedia Database Management Systems, Guojun Lu

Practical Guide to Software Quality Management, Second Edition, John W. Horch

Practical Insight into CMMI®, Tim Kasse

Practical Process Simulation Using Object-Oriented Techniques and C++, José Garrido

A Practitioner's Guide to Software Test Design, Lee Copeland

Risk-Based E-Business Testing, Paul Gerrard and Neil Thompson

Secure Messaging with PGP and S/MIME, Rolf Oppliger

Software Fault Tolerance Techniques and Implementation, Laura L. Pullum

Software Verification and Validation for Practitioners and Managers, Second Edition, Steven R. Rakitin

Strategic Software Production with Domain-Oriented Reuse, Paolo Predonzani, Giancarlo Succi, and Tullio Vernazza

Successful Evolution of Software Systems, Hongji Yang and Martin Ward

Systematic Process Improvement Using ISO 9001:2000 and CMMI®, Boris Mutafelija and Harvey Stromberg

Systematic Software Testing, Rick D. Craig and Stefan P. Jaskiel

Systems Modeling for Business Process Improvement, David Bustard, Peter Kawalek, and Mark Norris, editors

Testing and Quality Assurance for Component-Based Software, Jerry Zeyu Gao, H. -S. Jacob Tsao, and Ye Wu

User-Centered Information Design for Improved Software Usability, Pradeep Henry

Workflow Modeling: Tools for Process Improvement and Application Development, Alec Sharp and Patrick McDermott

For further information on these and other Artech House titles, including previously considered out-of-print books now available through our In-Print-Forever® (IPF®) program, contact:

Artech House
685 Canton Street
Norwood, MA 02062
Phone: 781-769-9750
Fax: 781-769-6334
e-mail: artech@artechhouse.com

Artech House
46 Gillingham Street
London SW1V 1AH UK
Phone: +44 (0)20 7596-8750
Fax: +44 (0)20 7630-0166
e-mail: artech-uk@artechhouse.com

Find us on the World Wide Web at:
www.artechhouse.com