MANAGING THE SYSTEM LIFE CYCLE

Second Edition

Edward Yourdon

MANAGING THE SYSTEM LIFE CYCLE

Second Edition

Edward Yourdon

YOURDON PRESS
A Prentice Hall Company
Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging-in-Publication Data

Yourdon, Edward.
Managing the system life cycle.

(Yourdon Press computing series)
Bibliography: p.
Includes index.
1. Computer software—Development—Management.
2. Electronic data processing—Structured techniques.
I. Title. II. Series.
QA76.76.D47 1988 005.1 87-23001
ISBN 0-13-547530-9

Editorial/production supervision
and interior design: Richard Woods and Barbara Marttine
Cover design: Wanda Lubelska Designs
Manufacturing buyer: Richard Washburn

The publisher offers discounts on this book when ordered in bulk quantities. For more information, write:

Special Sales/College Marketing
Prentice Hall
College Technical and Reference Division
Englewood Cliffs, NJ 07632

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-547530-9 025

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

Preface

Much has happened since the first edition of this book was published in 1982. This is inevitable in the computer field, of course, and I expect that significant changes will continue to occur through the rest of this decade and well into the 1990s.

This book is about the management of systems development projects that use such "structured" techniques as structured analysis, structured design, and structured programming. When I began writing the first edition of this book around 1980, structured programming was widely accepted and structured design was widely discussed, but structured analysis—probably the most important of the structured techniques—was not in common use at all. Now, in the late 1980s, all these structured techniques are well understood by computer programmers and systems analysts and have been used to help build thousands of computer systems around the world.

However, the structured techniques themselves have evolved and changed during this period, and those changes are reflected in this new edition. Probably the most important change is the incorporation of *data* modeling into a set of systems analysis techniques that previously concentrated almost exclusively on *function* modeling; today's system models include, along with the familiar data flow diagrams, various forms of *entity relationship diagrams,* which replace the data structured diagrams associated with the "classical" structured techniques of the late 1970s and early 1980s. Modern structured analysis also has tools for modeling the time-dependent behavior of systems (using *state transition diagrams,* sometimes known as "finite-state diagrams") and can thus be used on real-time systems as well as on the classical business-oriented systems.

There have also been changes in the "methodology" of structured analysis. The first edition of this book emphasized a series of analysis activities that led to four distinct models of a system: the "current physical," the "current logical," the "new logical," and the "new physical." Although there were good reasons for asking the analyst to build these models, our experience in the real-world use of these methods in hundreds of consulting projects during the past several years has shown us the danger of spending too much time modeling the user's current system—a system that will of necessity be thrown away and replaced. Consequently, this edition reflects the current view of structured analysis, in which the analyst is asked to begin with a model of what the user's *new* system must do.

Similarly, "classical" structured techniques of the late 1970s and early 1980s suggested that the model of user requirements could be transformed directly into a model of code organizations; this was because most systems built in those ancient times tended to be implemented as monolithic batch programs running on a single mainframe computer. This edition takes note of the fact that many systems are implemented on a network of computers and on a network of communicating processes (or "tasks" or "partitions," etc.) within each processor. Hence there are two intermediate design modeling activities (one at the processor level, and one at the task level) before the conventional structured design activity of producing structure charts is reached.

The concept of *prototyping* has also become popular in the past few years and is discussed in this book. Though prototyping tools are an important addition to the "tool kit" available to the project team, I continue to be concerned that prototyping is being sold (often by the vendors who created the software tools) as a panacea that will replace much of the work of structured analysis with instant gratification. I remain convinced that many of the benefits of prototyping can be achieved with the structured techniques; this is discussed in Chapter 3.

Finally, we have seen the advent of many *automated tools* for systems analysts and designers in the years since the first edition of this book was published. These tools, developed by such companies as Intech, Nastec, Yourdon, Cadre, and Tektronix, enable the systems analyst and systems designer to automate much of the graphics and text associated with the requirements models and implementation models of a system. One could argue that these tools do not change the project activities themselves or the way that the project activities are managed; indeed, the project team must carry out systems analysis whether or not there is automated support, and the project manager must ensure that the proper "products" of systems analysis are produced at the proper time. However, I am convinced that the *quality* of work produced by the systems analyst and designer will improve dramatically with the use of such tools, and this will (indirectly) improve the manageability of the project.

I have had the honor and good fortune to work with a number of excellent, dedicated data processing professionals during the years since the first edition of this book was published; many of them have offered insights and suggestions that have helped me tremendously in the preparation of this new edition. I am especially grateful to Bob Spurgeon, William Wenker, and Eric Blaustein for their careful reading of the draft manuscript and for their many helpful suggestions. Finally, I would like to thank my wife and children for their patience and forbearance while I sat hunched over my Macintosh computer for days on end; they deserve a great deal of credit for helping create this book.

Introduction

WHAT THIS BOOK IS ABOUT

The purpose of this book is to present a system life cycle for systems development projects that makes use of structured systems development techniques, in sufficient detail for you to use in a real project. In other words, this book is designed to give you the necessary guidelines with which to organize, manage, and control a systems development project—in particular one that makes use of structured analysis, structured design, and structured programming.

Why should you read this book? Presumably because you're looking for a way to organize your systems development projects or because you're not satisfied with the methodology that you currently use. By the end of this book, you will have learned the important *activities* and *products* associated with a systems development project, and you should be able to apply the concepts to your next project.

I also hope to shed some light on an issue that often generates loud, emotional debates in many organizations, the issue of "methodologies." Three questions arise: Should our organization use a methodology? If so, what should it look like? What should its components be?

A *methodology,* as we will use the term in this book, is a formal specification of a *system for building systems.* It defines the pieces, or components, of the system for building computerized information systems—that is, the phases or activities that one finds in a typical software development project. It also defines the interfaces between those components.

You'll also find that this book makes a distinction between a *technical* model of the system for building systems and a *managerial* model. Most methodology products and textbooks on project management concentrate entirely on the latter: They tell the manager how to control and supervise the activities in his project. But as my colleague Tom DeMarco points out, "You can't have a methodology without methods." You can't hope to organize, manage, and control a systems development project unless you can describe to your technicians—systems analysts, systems designers, database designers, and programmers—what you expect them to be doing during various phases of the project and what kind of technical products you expect them to deliver. This book concentrates on that aspect of the methodology and places considerably less emphasis on the classical control and supervisory aspects of project management.

Throughout this book, I assume that you have had some prior exposure to systems development projects, either as a programmer-analyst or as a project leader. I hope that this book will be useful as well to high-level managers, training coordinators, university professors, presidents, and kings—but in my mind's eye, it's the project leader responsible for project management whom I'm addressing as I write these words.

WHAT THIS BOOK IS NOT ALL ABOUT

There are several things that I am definitely *not* attempting to accomplish in this book.

- *This book will not teach you how to be a manager.* If you don't know how to manage and motivate human beings, don't expect any help from this book. If you don't have a basic sense of delegation, organization, and administration, I'm not sure you can obtain help from *any* book; but you certainly won't get any pearls of wisdom from this one! This is, of course, a major issue for the programmer or systems analyst who has just been promoted to the rank of project manager and is faced with his first project. If you are in this situation, consult standard introductory texts on projеct management before you read this book.
- *This book will not tell you how to organize your EDP or MIS department.* This book is concerned with the activities of an individual systems development project; as you'll see in Chapter 3 and the subsequent chapters, we will talk about the interactions between the project team and management end users (otherwise known as "clients," "customers," or "owners"), and the operations department. But there is little or nothing said about such things as a "steering committee" of users and managers who help set priorities among the various projects in the

organization. Nor is anything said about how the technicians in the EDP or MIS department should be organized for maximum efficiency—whether systems analysts and programmers should be separate groups or whether development teams should be kept separate from maintenance teams. However, the book will provide some comments on the impact of the structured systems development techniques on the typical "life cycle" that an EDP or MIS organization is likely to use.

- *This book will not make you an expert in using structured systems development techniques.* If you've never heard of structured programming or structured analysis, this is not the book to read. There are lots of good books that will give you the basics; the discussion in Chapter 2, for example, provides an overview and some references for additional reading. But for the most part, this book assumes that you're already familiar with the basic concepts of structured analysis, structured design, structured programming, top-down development, and walkthroughs.
- *This book will not make you an expert estimator.* I can't estimate how long it will take to build an underground house in a swamp—yet that's the sort of thing many of our end users and managers really want us to do! Chapter 10 discusses some of the problems of scheduling and estimating, but I should warn you in advance: *There is no magic.* For more information on this area, I recommend that you consult some of the classic books on software metrics, such as those by DeMarco (1982), Boehm (1982), and Jones (1986) in the bibliography.
- *This book will not help you solve political problems.* Virtually everything in this book assumes that you are a rational manager, that you have rational programmers and systems analysts working for you, that you deal with rational users and customers, and that the environment in which you determine your schedules and manpower estimates is not only rational but also friendly and supportive. I assume that you, your subordinates, your users, and your superiors are interested in producing a "quality" information system—one that does what the user wants, and does it in a reasonably economical, reliable, maintainable fashion. If your environment is just the opposite—if, for example, you are rewarded *solely* for finishing your project on time, with no concern whatsoever about the quality of the product—this book probably won't help you. If you have arbitrary and unreasonable deadlines imposed on you from on high, or if you have users who refuse, on principle, to discuss their requirements with you, you're in deep trouble. Rather than reading this book, I suggest that you consider polishing your résumé and looking for a better place to work. As an alternative, consult such books at Block's *Politics of Projects* (1983) or Page-Jones's *Practical Project Management* (1985) for advice and guidance.
- *This book will not provide you with a "religious" view of software*

development. Many textbooks on project management, and most software development methodology packages, take a hard-line approach to the various activities that they identify in the project life cycle—that is, they emphasize that the programmer-analyst *must* carry out tasks A, B, and C in a certain sequence in *all* cases, under *all* conditions. That's understandable in the case of methodology products, whether purchased for $50,000 or developed within the corporation for a cost sometimes approaching $500,000. When that kind of money is involved, the person who commits to the expenditure usually defends the package the same way a wild animal defends its offspring against predators: "If you spend all that money on the XYZ methodology package," says the corporate EDP manager, "you can be damned sure that every single one of my programmers will memorize every page of it and follow it to the letter!" Considering the relatively low cost of this book, there should be no need for religious mania—even if you discard 90 percent of the ideas in the book, it won't have cost you $500,000. Indeed, I will deliberately suggest that you improvise in certain areas and that you fill in the details in other areas based on your own experience, and I hope that you feel equally free to ignore sections of this book that don't apply to your own needs.

THE ORGANIZATION OF THIS BOOK

Now that I've given you a thumbnail sketch of what the book will and will not provide, let me give you a slightly more detailed picture of the individual chapters.

Chapters 1 and 2 serve as an introduction to the main subject. Chapter 1 compares the advantages and disadvantages of conventional systems development projects, semistructured projects, and modern structured projects. Chapter 2 contains a brief overview of the structured techniques themselves, with suggested references in case you need more detail. If you're already familiar with the structured techniques, and if you're feeling eager to proceed, turn directly to Chapter 3.

The heart of the book consists of seven chapters, 3 through 9. Chapter 3 gives an overview of the structured project life cycle and its component activities. Chapters 4 through 8 discuss the major activities: survey, analysis, design, and implementation. Chapter 9 addresses the final major activities, including generation of user manuals, conversion of the database, installation of the system, and quality assurance.

The concluding chapter sums it all up, placing emphasis on the manager's special role and obligations.

Contents

1

Changes in Project Management

1.1 INTRODUCTION

As I indicated in the introductory chapter, this book is concerned with a relatively new kind of life cycle for systems development projects—the *structured project life cycle.* It is also concerned with the proper use of structured analysis, structured design, and structured programming in a systems development project. Consequently, there is a strong implication that the new, structured kind of life cycle is in some way better than the conventional project life cycle. And, similarly, that the use of structured analysis and related techniques leads to far more successful projects than does the use of conventional analysis and development techniques. Indeed, one of the premises of this book is that the conventional projects tend to be over budget, behind schedule, expensive to develop, expensive to maintain, unreliable, and unacceptable to users.

Of course, these problems do not apply to *all* conventional EDP projects, but the larger the scope and size of the project, the more likely it is to show evidence of some or all the difficulties just listed. I think it is reasonable to make the following observations about the relationship between the size of a project and its complexity:

SIZE OF PROJECT	LEVEL OF COMPLEXITY
Up to 1,000 lines	Trivial
1,000 to 10,000 lines	Simple
10,001 to 100,000 lines	Difficult
100,001 to 1,000,000 lines	Complex
1,000,001 to 10,000,000 lines	Nearly impossible
More than 10,000,000 lines	Utterly absurd

(Obviously, lines of code is only one measure of complexity, and a primitive one at best. However, it is adequate to illustrate my point in this chapter.)

Trivial projects, defined in terms of lines of code, may not require any formal project management. A computer program or system that involves only a few hundred lines of code can usually be implemented by one person in a period of a few days to a few weeks. Generally, all one wants to know at the beginning of such projects is the deadline: "When is it going to be finished?" And even though structured analysis, structured design, and structured programming can be of great benefit even in such small projects, the programmer—by brute force or just common sense—can get the job done using conventional methods. If problems arise and the schedule slips, the manager can always fall back on what my colleague Tom DeMarco calls "the time-honored tradition of unpaid overtime": The manager can gently persuade the programmer to work nights and weekends until the project is finished.

Indeed, much the same thing can be said about *simple* projects, those involving up to 10,000 lines of code. Such a project typically involves three or four programmer-analysts for a period of 6 to 12 months. It's large enough, and it lasts long enough, for some kind of formal project management to be necessary, but it's the sort of project that a veteran manager has experienced dozens of times in his career. Precisely because it *is* within the manager's realm of comprehension, he is able to organize, manage, and control it. And although the project life cycle specified in this book is quite useful, managers will argue that implementing such a formal discipline represents unnecessarily heavy artillery. Similarly, although the techniques of structured analysis, structured design, and structured programming will lead to a more maintainable product, their use may not speed up development time at all.[1]

However, projects involving between 10,001 and 100,000 lines of code—projects that I categorize as *difficult*—are almost beyond the ability of the manager to handle easily. Such a project can involve half a dozen to a dozen programmers and can last two or three years. Formal project organization

[1] In general, proper use of the structured techniques will improve the productivity of the development phase of a system by approximately 10 to 15 percent. However, maintenance costs are usually reduced by a very substantial amount, usually ranging from a factor of 2 to a factor of 10.

is an obvious necessity, and a formal approach to the analysis and design of the EDP system is also usually essential. In most medium to large EDP organizations, a project of this size will eventually finish—if only because the organization continues to pour people and money into the project until it staggers through to completion.[2] However, there is a nontrivial danger that the project will be finished several months late and that it will exceed its budget by tens of thousands or even hundreds of thousands of dollars. Consequently, the techniques of structured analysis, design, and programming become critically important; with the proper use of these techniques, there is a reasonable chance that the project can be completed on time and within the stated budget.

Perhaps more important than use of the techniques is the use of the *structured project life cycle* in projects of this size. A project involving up to 100,000 lines of code is sufficiently complex that neither the systems analyst nor the user is likely to have a crystal-clear understanding of exactly what the system is supposed to do, and even if there *is* a clear understanding, the user is likely to change his mind about some of the requirements. Also, if the project lasts two or three years, there is a reasonable chance that the environment—that is, the technology, the local business conditions, the applicable government regulations, and even the user himself—will change by the time the project is finished. Nevertheless, a veteran project manager may tell you that none of these problems is insurmountable, that it's all a question of "good management."

With *complex* projects, involving between 100,001 and 1,000,000 lines of code, even the most battle-scarred veteran would admit to a certain amount of nervousness. At this level of complexity, 50 to 100 people may be involved in a project that can last three to five years or more. On such a project, many members of the project team either will leave before it is finished or will be hired in the middle, and there will be at least two levels of project management. In addition, we can be reasonably sure that the system is not being built for just one user or even one homogeneous group of users. Instead, there are likely to be diverse (and often conflicting) communities of users, each of which has its own local view of the requirements of the system.

If conventional development techniques are used in a project life cycle, there is a good chance that (a) the project will never be finished, (b) it will be finished so far behind schedule and so much over budget that it will damage or ruin the manager's career, (c) the user will reject the final system as being unsuited to his needs, or (d) all of the above. Although the techniques espoused

[2] However, in today's turbulent business world, the user may not be willing to continue spending money on a project once it gets substantially behind schedule or over budget. Various surveys, including one by T. Capers Jones (1986), have reported that as many as 25 percent of all DP projects never finish!