MANAGEMENT METHODOLOG FOR SOFTWARE PRODUCT ENGINEERING

RICHARD C. GUNTHER

MANAGEMENT METHODOLOGY FOR SOFTWARE PRODUCT ENGINEERING

RICHARD C. GUNTHER
Manager
Product Planning
and Market Analysis
Amdahl Corporation

A Wiley-Interscience Publication
JOHN WILEY & SONS
New York Chichester Brisbane Toronto

Library of Congress Cataloging in Publication Data:

Gunther, Richard C., 1937–
Management methodology for software product engineering.

"A Wiley-Interscience publication."
Bibliography: p.
Includes index.
1. Computer programming management. I. Title.

QA76.6.G85 658'.05 78-711
ISBN 0-471-33600-9

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

PREFACE

Back in 1620 Francis Bacon challenged the Aristotelian view of the sciences as unrelated collections of aphorisms. In his *New Organon* he introduced a method of induction whereby he iterated from observed facts to derived axioms to further experiments and then to new axioms. Through this method he discovered underlying principles common to many sciences. Simultaneously, René Descartes codified the procedures of science into self-consistent systems of theory.

We who call ourselves software engineers have taken a similar approach. We challenge the view that the production of software is fundamentally different from the production of anything else. We observe the facts accumulated over several years of software production, compare them with other facts from our environment, and exploit similarities to determine underlying principles. While we still share Bacon's uncertainty about the exact character of abstractions, our experience shows us that there do exist general principles, and that adherence to them improves our ability to produce software on time, within budget, and according to specification.

In this book you will find a methodology for managing the planning, design, construction, evaluation, documentation, distribution, and maintenance of software. This methodology exploits the high correlation between phases in the life cycle of a software system and the functions that must be performed throughout the cycle. You will see how top-down design, management by objectives, configuration management, and other principles can be integrated through a self-consistent set of documents and procedures that continually reinforce one another.

This book emphasizes the product concept; the concept of heavy-duty software—software to be used by a vast, perhaps little known body of users and to be promoted, maintained, and enhanced over a long period of time. It presents techniques and tools tailored for managing such heavy-duty software. You can select from among them those that meet your needs, whether you are a data processing

manager, a defense system project leader, a software company product manager, or a director of software development for a hardware manufacturer. The methodology presented is fully compatible with structured programming and with chief programmer teams, but it works equally well with or without them.

The book is divided into five parts: background, techniques, tools, appendix, and references. In the first part you are introduced to concepts that are fundamental to the rest of the book: software as a product, the life cycle, the phase-function matrix, hierarchical decomposition. A sample company and a sample software product are introduced. This company and product are featured throughout the book to demonstrate how highly correlated the concepts are and how self-consistent the documents and procedures are. In the second part each function involved in software production is discussed separately. The role of each function in every phase of a product's life cycle is explored. This organization enhances the book's value for reference: you can easily review a single function or you can review a single phase by studying a selected section of each chapter. The third part discusses in great detail the semantics and syntax you will need to implement the book's methodology in self-consistent plans, specifications, and reports. Particular emphasis is placed on an integrated set of design documents that observe a rigorous decomposition of substance and by life cycle phase. Finally, the fourth part includes a complete example of the master plan for development of a software product and the fifth part has a list of references you can use to amplify the text.

Every technique and tool presented in this book has been successfully employed somewhere and many can be employed independently of one another. They are fully compatible with structured programming and chief programmer team concepts. They are based on top-down, modular design. Because they are tightly correlated, the more you employ them together the more you will benefit from synergism as they reinforce one another. Where necessary, you are cautioned to avoid introducing critical elements too soon or too late or too fast for assimilation.

Why should you read *Management Methodology for Software Product Engineering* instead of or in addition to any of a number of other books on the management of programming? First, because you want to produce heavy-duty software and because of that you have complex communication problems to overcome which texts without a product orientation fail to acknowledge. Second, because you want assurance that as you adopt each technique or tool, it will still work

after you introduce other techniques and tools. Not only will it continue to work; it will work better. Lastly, because you want to adopt a methodology that is founded on sound planning, documentation, and review principles that assure at the beginning of a development project you will not overlook something and have to take costly corrective action in midstream.

This book is the result of many years of evolution, beginning with the early work of Kenneth W. Kolence, who first kindled my interest in software engineering management and later inspired me to undertake the task of writing the book. As a practitioner of the subject I found it difficult to make time available for the project, and I owe thanks to Clair E. Miller and Kornel Spiro for their understanding and encouragement over the years it took to complete the work. Most of all I am indebted to my family, who allowed me many evenings and weekends to work, and especially to my wife Suzanne, who more than anyone else made it possible for me to finish.

RICHARD C. GUNTHER

Palo Alto, California
January 1978

CONTENTS

Chapter
1

About This Book

The title of this book, *Management Methodology for Software Product Engineering,* suggests that it is a book about managing software engineering—which it is. But what is software engineering? The term originated with, or at least was popularized by, the two conferences sponsored by the North Atlantic Treaty Organization in 1968 and 1969 (1). The science of **software engineering** has progressed since then to the point where now, according to Yeh (2), it is an engineering approach to computer software development encompassing programming methodology, software reliability, performance and design evaluations, software project management, and program development tools and standards.

1.1 WHAT THIS BOOK IS AND IS NOT

While this book does not claim to cover all of the above mentioned topics, it does provide planning and control techniques and tools that discourage poor and encourage good programming methodology and software reliability, ensure comprehensive and timely design evaluations, and improve and simplify project management. Thus it does provide an engineering approach to software reliability, design evaluation, and project management. It does not address itself to programming methodology, performance evaluation, or program development tools and standards except to make occasional recommendations and to point to appropriate references for more information.

This is a book about software products and begins with the assumption that there is a difference between a computer program or a system of computer programs and a software product. A **software product** is a computer program plus all of the planning, documentation, testing, publications, training, distribution, maintenance, and control that comprise the aggregate heavy-duty software—software to

be installed at more than one site, for use by people not known by the developers, in ways not anticipated by the developers. Many good books and articles have been written about the development of software systems (not software products) for an audience that is the institutional or corporate data processing or information systems department. Such works make one or more of the following assumptions:

- The developer is the user or is at least organizationally related to the user.
- The user specifies his requirements directly to the developer.
- The user participates in design reviews.
- The software must run on only one or on a very limited range of hardware configurations.
- The developer installs the software for the user.
- Problems in using the software are resolved by direct interaction between the user and the developer/maintainer.

Except in the limited case where it is developed on contract for a single customer, none of the above assumptions is likely to apply to a software product. Therefore, this book makes assumptions that are essentially contrary to those above:

- The developer is unacquainted with the user.
- User requirements either are deduced by the developer or are presented to him by an intermediary, such as a marketing support organization.
- Users do not participate in design reviews, except possibly when represented by an intermediary.
- The software must run on a wide range of hardware configurations, in a wide range of software environments.
- Users install the software themselves or have someone other than the developer do it for them.
- Problems are resolved by correspondence, sometimes through an intermediary.

This book, then, is written mainly for an audience interested in providing and maintaining software for multiple and diverse users who are continually at arms' length from the developers and maintainers. All of these assumptions apply to computer manufacturers and

software vendors. Although there is close and frequent interaction between providers and users of software systems, institutions and corporations with widely distributed computer processing requirements face many of the distribution, reliability, and control problems addressed in this book. The data processing manager who serves a limited, well-known user community can also benefit from this book. The more of its systems he employs, the better he will serve his users and the easier it will be for him to expand his services, adding one system after another.

This book, then, is about software products. About what else is it or is it not? Well, it is a book on management, not on programming. It is a book on software engineering, but on how to manage it rather than on how to practice it. Again, there are many excellent works on programming, documenting, and testing. There is no attempt to duplicate them here, although several of them are referenced so that you can find information that is consistent with the management systems presented here. Consistency is emphasized because it is a key concept in this book. The book is a collection of management principles, concepts, and practices that have been proven to work somewhere at some time. But it is much more because each idea embodied in the book has been designed or reworked to fit an overall grand design in which all of the tools and techniques reinforce one another. Each can be taken from the book and used by itself, but if they are taken all together they synergistically produce a system far more productive than the sum of the parts.

In a thought-provoking article (3), Nolan hypothesizes a stage theory for managing computer resources. A stage theory premises that a system—economic, sociological, galactic—evolves through distinct stages that can be abstracted into a taxonomy. Such a stage theory appears to apply to the way the production of heavy-duty software is evolving. If Mr. Nolan were to study the methodology of heavy-duty software production as he studied the use of computers, he might derive a chart like the one shown in Figure 1.1. This figure hypothesizes three stages: the Age of Programming, the Age of Software Development, and the Age of Software Engineering. While an observer of the computer industry might conclude that the industry as a whole traversed these stages in the periods indicated in Figure 1.1, he would, observing one heavy-duty software supplier at a time, see each supplier traverse the three stages in sequence no matter when the supplier entered the first stage. Said another way, a stage theory predicts that if the theory applies to a system, it applies to each member of the system. An objective of this book is to shorten

Attribute	Age of Programming	Age of Software Development	Age of Software Engineering
Principal period	1950–1962	1963–1971	1972–?
What is generated	Computer programs	Software	Software products
Generation or selection sequence	Hardware first	Parallel	Software first
Construction method	Monolithic	Modular	Structured
Design process	Bottom-up	Top-down	Top-down
Construction process	Bottom-up	Bottom-up	Top-down
Quality assurance	None	By developer	Independent
Production organization	Individual effort	Project team	Chief programmer team
Maintainability and reliability	Not considered	Afterthought	Designed-in
Maintenance	None	By developer	By specialists
Users	Few, sophisticated	Many, initiated	Very many, uninitiated
Use of development support libraries	None	Occasional	Mandatory
Language used	Assembly	Assembly, higher-level	Higher-level implementation
Design documentation telling			
Why?	None	None	Before construction
What?	Before construction	Before construction	Before construction
How?	None	After construction	During construction
Project control	None	Critical path methods	Critical path methods and management by objectives

Figure 1.1 A possible stage hypothesis for heavy-duty software development.

the time for you, as a supplier of heavy-duty software, to get from the Age of Programming to the Age of Software Engineering by describing and rationalizing many attributes of software engineering. While following this book's methodology is not a necessary condition for reaching the Age of Software Engineering, it is a sufficient condition.

A prominent secondary theme in this book is project management. Organization of the development function into projects is now so widely accepted that it is taken for granted here. The concept of chief programmer teams that became popular in the early 1970s is encompassed, but is not a prerequisite. So, in addition to being a general treatise on software product management, this book is a book on project management.

1.2 WHERE TO LOOK FOR INFORMATION NOT IN THIS BOOK

As noted previously, there are many writings on managing software development and on computer programming. Few on either subject deal with heavy-duty software, but the percentage of new writings on these subjects that does deal with heavy-duty software is increasing. Assuming the trend continues, watch for new titles to appear in the software engineering series of technical publishers. Watch also the proceedings of the National Computer Conferences sponsored by the American Federation of Information Processing Societies, and conferences sponsored by individual societies such as the Association for Computing Machinery and the Institute of Electrical and Electronics Engineers (IEEE). The conferences on software reliability sponsored by IEEE are particularly good. These societies are also setting up special interest groups on software engineering that publish papers and sponsor meetings. Organizations like the American Management Associations are becoming more active in software engineering. Project management and consulting firms in particular, are presenting training courses that emphasize the integration of many of the concepts set forth here. University curricula are still weak on software engineering; but indeed, according to Mills (4), "universities have no experience in even knowing what to teach." Structured programming and reliability engineering are becoming more popular in degree programs, and extension programs now offer courses in management. Teaching management in extension courses probably will continue to predominate, since the time when such training is most useful to the recipient is not while he is pursuing a